uc3m | Universidad Carlos III de Madrid

Bachelor's Degree in Computer Science & Engineering
2017/2018

Bachelor Thesis

Differential Privacy in the browser: Collecting data in Firefox

---

Alejandro Rodríguez Salamanca

Supervisor
Nerea Luis Mingueza
Leganés, March 6th 2018

Contents

List of Tables

List of Figures

Listings

Abstract

We live in a society in which knowledge is power. For the companies, knowing their users, how they behave, and what they like can help them to improve their products, show personalized ads, or, in the worst case, sell this data to third parties.

As a user, most of the time we are not aware of how much personal data we share and we do not know who can be collecting this data and how it is being used.

For this reason, it is necessary that the user data collected is anonymized. In this way, companies can continue obtaining information but users are not harmed. Here is where Differential Privacy appears. As we will see, this technique allows data collection and anonymization without compromising user's privacy. If an attacker intercepts the data, identifying a single user will not be possible regardless of the knowledge of the attacker.

This technique is very powerful, but the applications outside of the academic world are limited.

RAPPOR is an example of a real world application. It is an algorithm developed by Google to collect data applying local Differential Privacy, this is, anonymizing the data before sending it for analysis.

In this work it is explained the transition from the academic explanation of RAPPOR, performing experiments following the scientific method to validate its functioning, to an implementation in one of the most used web browsers in the world, Firefox, using software engineering techniques to plan, design and implement the project securing a correct design of the involved components. Moreover, a research on the legal and socioeconomic implications of this project is included.

Resumen

Vivimos en un mundo en el que el conocimiento es poder. Y para las empresas, conocer a sus usuarios, saber sus gustos y sus comportamientos, puede ayudarles a mejorar sus productos, mostrar publicidad más personalizada, o en el peor de los casos, vender dichos datos a terceros.

Como usuarios, muchas veces no somos conscientes de la cantidad de datos que generamos, y no sabemos quién puede estar recogiendo esos datos ni cómo los puede usar.

Por esto, es necesario que los datos que se recogen acerca de los usuarios estén anonimizados. De esta forma, las empresas pueden seguir obteniendo información, pero los usuarios no se ven perjudicados. Aquí es donde entra la privacidad diferencial. Como veremos, esta técnica permite recoger datos y anonimizarlos de tal forma que si un atacante los intercepta, nunca podrá identificar si los datos de cierto usuario están presentes, independientemente de la información que se posea de éste en el conjunto de datos.

Esta técnica es muy potente, pero no es posible encontrar muchas aplicaciones que se hayan puesto en práctica más allá del mundo académico.

Una de estas implementaciones es RAPPOR, un algoritmo desarrollado por Google para recoger datos aplicando privacidad diferencial local, esto es, anonimizando los datos antes de ser enviados para su análisis.

En este trabajo se recoge cómo se ha pasado de la explicación académica de RAPPOR, realizando experimentos siguiendo el método científico para validar su funcionamiento, a la implementación en uno de los navegadores más usados del mundo, Firefox, usando técnicas de ingeniería de software para planificar, diseñar y llevar a cabo el proyecto, asegurando un correcto diseño de los componentes involucrados. Además, se incluye un estudio de las posibles implicaciones legales y socioeconómicas que tiene este proyecto.

Acknowledgements

Gracias a mi familia, por el apoyo que he recibido durante toda mi educación. Lo hemos conseguido.

Gracias a mis amigos, sin vuestra ayuda no hubiese sido posible.

Gracias a Nerea, por supervisar este trabajo y confiar en mis decisiones.

Danke to all the people in Mozilla Berlin.

1 Introduction

Every year, computers have faster and cheaper memory and its performance increases (Moore's Law [1]). Moreover, Cloud Computing, which allows us to use computing services over the internet, is a reality and it offers the possibility of scaling our systems adding more machines easily. With all this capacity, companies have started to collect more data from their users to understand how they behave and improve their products, or to sell them to third parties.

Users have now to trust in third parties to store their data and as a consequence they are exposed to data breaches [2][3] that compromise its security and privacy.

In the mid 90s, The Massachusetts Group Insurance Commission (GIC) released anonymized data on state employees that showed every hospital visit. The goal was to provide data to researchers, and the state spent time removing all obvious identifiers such as name, address and Social Security number. A graduate student started hunting for the Governor's hospital records in the GIC data. She knew that Governor Weld resided in Cambridge, Massachusetts, a city of 54,000 residents and seven ZIP codes. For twenty dollars, she purchased the complete voter rolls from the city of Cambridge. This is a database containing, among other things, the name, address, ZIP code, birth date, and sex of every voter. By combining this data with the GIC records, she found Governor Weld with ease. Only six people in Cambridge shared his birth date, only three of them were men, and of them, only he lived in his ZIP code. The Governor's health records were de-anonymized. [4]

In 2007 Netflix offered a $1 million prize for a 10% improvement in its recommendation system. They also released a training data set for the competing developers to train their systems. In order to protect their customer's privacy, they removed personal information and replaced IDs with random IDs. However, Netflix is not the only movie-rating portal out there, there are many others such as IMDb. Researchers linked the Netflix data set with IMDb to de-anonymize the Netflix data set using the dates on which an user rated certain movies. [5]

Several techniques to anonimyze data keeping it useful for analysis have been proposed, such as k-anonymity [4], t-closeness [6] and l-diversity [7] but all of them are prone to attacks. Due to this, attackers can still make inferences about data sets that may harm individuals.

Differential Privacy [8] [9] is a promising technique that claims to preserve users privacy while remaining data useful. It is based on the idea of randomized response. Based on this, Google developed RAPPOR [10], an algorithm to collect anonymous data.

RAPPOR can be used to collect browsing data. This is an interesting application which can offer browser companies important data to optimize for certain websites and also identifying potential malware if the user's homepage is changed to a non-trusted source.

This approach is also useful for an organization like Mozilla, as it tries to balance the necessity of collecting data to improve Firefox and protect their users, while preserving the anonymity and privacy of those who use Firefox.

With this idea in mind, the efficacy of the algorithm is evaluated performing simulations, and a Firefox extension is developed to collect the user's homepage in a differential private manner.

This document is divided in different sections. First, we explain the current State of the Art in privacy preserving techniques. Next, we explain how RAPPOR works. Later, we show the results of the performed simulations and its conclusions. After this, we explain how the Firefox extension to collect the user's homepage has been developed. Finally, we discuss the software engineering principles applied to this project, the socioeconomic and legal implications, and the future work on this field.

The motivation of this work is to demonstrate that privacy preserving techniques such as differential privacy can be applied in a product used by millions of people.

The main objectives of this thesis are to show that differential privacy is a step forward in privacy preserving techniques, and to build a solution that can be used to collect data using this approach in Firefox.

2 State of the Art

Before explaining the core of this thesis, the current state of the art should be exposed to understand the previous and current techniques used to anonymize data.

First, an overview of anonymizing techniques is summed up to understand the problem and the solutions that have been proposed over time, its advantages and drawbacks. Then, the concept of differential privacy is introduced. Finally, we discuss different implementations for differential privacy.

2.1 Anonymizing data

The problem of anonymizing data while trying to keep it useful has been present even before the popularization of personal computers. Different techniques have been introduced to tackle this problem with the purpose of ensuring privacy for the users.

2.1.1 k-anonymity

k-anonymity [4] was introduced as an attempt to solve the following problem: "Given person-specific field-structured data, produce a release of a data with scientific guarantees that the individuals who are the subjects of the data cannot be re-identified while the data remain practically useful."

Sweeney showed that the triple (date of birth, gender, zip code) is sufficient to uniquely identify at least 87% of US citizens in publicly available databases [11]. Then leaving out any unique identifiers like name and Social Security Number is not enough, which would mean that this approach will never work regarding anonymity and privacy.

When applying *k-anonymity*, attributes are suppressed or generalized until the information for each person contained in the release cannot be distinguished from at least $k - 1$ individuals whose information also appear in the release. Thus prevents definite database linkages. At worst, the data released narrows down an individual entry to a group of k individuals.

For example, if you want to identify a person and the only information you have is gender and zip code there should be at least k number of people meeting the requirement.

A way of k-anonymize data is using the suppression technique.

Suppression
This technique can replace individual attributes with an asterisk (*).

Table 1 be 2-anonymize as follows:

Name	Surname	Age	Race
Harry	Stone	34	African
Carlos	Simons	36	Caucasian
Linda	Stone	34	African
Carlos	Sanchez	22	Hispanic

Table 1: Data before using *k-anonymity*

Name	Surname	Age	Race
*	Stone	34	African
Carlos	*	*	*
*	Stone	34	African
Carlos	*	*	*

Table 2: Data after using *k-anonymity*

In such a way that rows 1 and 3 are identical and Rows 2 and 4 are identical.

The cost of k-anonymous solution to a database is the number of *'s introduced. The problem here is that it is possible to guarantee *k-anonymity* substituting every cell by *, but this renders the database useless. On the other hand, not enough cells removed would result in a weak database.

The suppression technique can be combined with generalization, replacing individual attributes with a broader set of categories, to increase the anonymity of the database.

Name	Age	Gender	State of domicile	Religion	Disease
Ramsha	29	Female	Tamil Nadu	Hindu	Cancer
Yadu	24	Female	Kerala	Hindu	Viral infection
Salima	28	Female	Tamil Nadu	Muslim	TB
Sunny	27	Male	Karnataka	Parsi	No illness
Joan	24	Female	Kerala	Christian	Heart-related
Bahuksana	23	Male	Karnataka	Buddhist	TB
Rambha	19	Male	Kerala	Hindu	Cancer
Kishor	29	Male	Karnataka	Hindu	Heart-related
Johnson	17	Male	Kerala	Christian	Heart-related
John	19	Male	Kerala	Christian	Viral infection

Table 3: Data before using *k-anonymity*

Because k-anonymization does not include any randomization, it is still possible to infer data from individuals given some knowledge about the database. For example, if the 19-year-old John from Kerala is known to be in the database above, then it can be reliably said that he has either cancer, a heart-related disease, or a viral infection. These attacks are even more powerful when several data sets can be correlated. Moreover, adding, changing or removing tuples may compromise *k-anonymity* if the database is not k-anonymize again.

Name	Age	Gender	State of domicile	Religion	Disease
*	$20 < \text{Age} \leq 30$	Female	Tamil Nadu	*	Cancer
*	$20 < \text{Age} \leq 30$	Female	Kerala	*	Viral infection
*	$20 < \text{Age} \leq 30$	Female	Tamil Nadu	*	TB
*	$20 < \text{Age} \leq 30$	Male	Karnataka	*	No illness
*	$20 < \text{Age} \leq 30$	Female	Kerala	*	Heart-related
*	$20 < \text{Age} \leq 30$	Male	Karnataka	*	TB
*	$\text{Age} \leq 20$	Male	Kerala	*	Cancer
*	$20 < \text{Age} \leq 30$	Male	Karnataka	*	Heart-related
*	$\text{Age} \leq 20$	Male	Kerala	*	Heart-related
*	$\text{Age} \leq 20$	Male	Kerala	*	Viral infection

Table 4: Data after using *k-anonymity*

Simply outputting every record k times will satisfy *k-anonymity*.

Meyerson and Williams [12] demonstrated that optimal *k-anonymity* is a NP-hard problem, and although there are heuristic methods that often yields practical results, they are not optimal.

2.1.2 l-diversity

We have seen that *k-anonymity* is susceptible to many attacks. *l-diversity* tries to solve this problem. The book Privacy-Preserving Data Mining – Models and Algorithms (2008) [7] defines *l-diversity* as being:

> Let a q*-block be a set of tuples such that its non-sensitive values generalize to q*. A q*-block is l-diverse if it contains l "well represented" values for the sensitive attribute S. A table is l-diverse, if every q*-block in it is l-diverse.

The paper *t-closeness: Privacy beyond k-anonymity and l-diversity* [6] defines *l-diversity* as being:

> An equivalence class is said to have l-diversity if there are at least l "well-represented" values for the sensitive attribute. A table is said to have l-diversity if every equivalence class of the table has l-diversity.

2.1.3 t-closeness

t-closeness is a further refinement of *l-diversity* group based anonymization that is used to preserve privacy in data sets by reducing the granularity of a data representation.

The original paper by Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian [6] defines *t-closeness* as:

> An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness.

2.2 Differential Privacy

Differential privacy [8] formalizes the idea that a query should not reveal whether any person is present in a data set, much less what their data is.

A simple example used in social sciences is to ask a person to answer the question "Do you own the attribute A?", according to the following procedure:

1. Throw a coin.
2. If head, then answer honestly.
3. If tail, then throw the coin again and answer "Yes" if head, "No" if tail.

If the attribute A is synonymous with illegal behavior, then answering "Yes" is not incriminating but, overall, these data with many responses are significant, since positive responses are given to a quarter by people who do not have the attribute A and three-quarters by people who actually possess it. Thus, if p is the true proportion of people with A, then we expect to obtain:

$$(1/4)(1-p) + (3/4)p = (1/4) + p/2$$

positive responses. Hence, it is possible to estimate p.

As Dwork says in the book *The Algorithm Foundation of Differential Privacy, 2013*, the concerns with other approaches to anonymize data are:

- *Data Cannot be Fully Anonymized and Remain Useful*: The richer the data, the more interesting and useful it is. However, the richness of the data enables "naming" an individual by a sometimes surprising collection of fields, or attributes, such as the combination of zip code, date of birth, and sex, or even the names of three movies and the approximate dates on which an individual watched these movies. This "naming" capability can be used in a linkage attack to match "anonymized" records with non-anonymized records in a different dataset.
- *Re-identification of "Anonymized" Records is Not the Only Risk*: Re-identification of anonymized data records is clearly undesirable, because the record may contain compromising information that could cause harm.
- *Queries Over Large Sets are Not Protective*: Questions about specific individuals cannot be safely answered with accuracy. Forcing queries to be over large sets is not a panacea, as shown by the following differencing attack. Suppose it is known that Mr. X is in a certain medical database. Taken together, the answers to the two large queries "How many people in the database have the sickle cell trait?" and "How many people, not named

X, in the database have the sickle cell trait?" yield the sickle cell status of Mr. X.

- *Query Auditing Is Problematic*: One might be tempted to audit the sequence of queries and responses, with the goal of interdicting any response if, in light of the history, answering the current query would compromise privacy. The auditor may be on the lookout for pairs of queries that would constitute a differencing attack. There are two difficulties with this approach: First, it is possible that refusing to answer a query is itself disclosive. Second, query auditing can be computationally infeasible.
- *Summary Statistics are Not "Safe"*: the failure of summary statistics as a privacy solution concept is immediate from the differencing attack just described.
- *"Ordinary" Facts are Not "OK"*: Revealing "ordinary" facts, such as purchasing bread, may be problematic if a data subject is followed over time.
- *"Just a Few"*: In some cases a particular technique may in fact provide privacy protection for "typical" members of a data set, or more generally, "most" members. In such cases one often hears the argument that the technique is adequate, as it compromises the privacy of "just a few" participants.

To continue talking about differential privacy, some concepts should be defined first [13]:

- Dataset: A Dataset d is a collection of records from a Universe U. If we were collecting data about flip coins of three individuals, we would have the universe $U = head, tail$ and our dataset d would have two entries, $d_{head} = i$ and $d_{tail} = j$, where $i + j = 3$.
- Distance: The distance measures how many records differ between two databases. In the previous example, the distance between two different datasets x and y can be defined with the l_1 norm:

$$||x - y||_1 = \sum_{i=1}^{|U|} |x_i - y_i|$$

- Mechanism: A mechanism is an algorithm that takes a dataset as input and returns an output. If mechanism C counts the number of individuals in the dataset, then $C(d) = 3$ in this example. In practice we will consider randomized mechanisms, where the randomization is used to add privacy protection.

Now, we can define Differential Privacy as:

Theorem 1 *A mechanism M satisfies ϵ-differential privacy if for every pair of datasets x, y such that the distance between x and y, $||x - y||_1 \leq 1$ and for every*

subset $S \subseteq Range(M)$ *[8]:*

$$\frac{Pr[M(x) \in S]}{Pr[M(y) \in S]} \leq e^{\epsilon}$$

It is important to clarify that this is just a definition, not an algorithm. That means that it is just a condition that must be satisfied by a mechanism to claim that it satisfies ϵ differential privacy.

Let's check if the previous example satisfies 1-differential privacy. As a reminder, we were given a mechanism C, counting the number of individuals in a dataset composed by data about flip coins of three individuals.

Given the previous definition, we would need to prove that C satisfies:

$$\frac{Pr[C(x) \in S]}{Pr[C(y) \in S]} \leq e^{1}$$

In this case, it is easy to find a counter example. Given x, y such that the distance $||x - y||_1 = 1$ (the dataset x contains one more record than the dataset y) and $||x||_1 = k$ (the dataset x contains k records), then:

$$\frac{Pr[C(x) = k]}{Pr[C(y) = k]} \leq e \Rightarrow \frac{1}{0} \leq e$$

As the probability of $C(x)$ containing k records is 1, and therefore the probability of $C(y)$ containing k records is 0.

According to Dwork [8], the key qualities of Differential Privacy are:

1. Protection against arbitrary risks.
2. Automatic neutralization of linkage attacks.
3. Quantification of privacy loss. It is possible to measure the privacy loss in differential privacy and compare different techniques to determine which one provides better accuracy or better privacy.
4. Composition. Mechanisms can easily be composed assuming that the mechanisms operate independently given the data. Let d be a dataset and g an arbitrary function. Then, the sequential composition theorem asserts that if $M_I(d)$ is ϵ_i differentially private, then $M(d) = g(M_1(d), M_2(d), ..., M_N(d))$ is $\sum_{i=1}^{N} \epsilon_i$ differentially private.
5. Group Privacy.
6. Closure Under Post-Processing. Differential Privacy is immune to post-processing: A data analysis without additional knowledge about the private database cannot compute a function of the output of a differential private algorithm and make it less differential private.

Now that we have a notion of what Differential Privacy means and what it tries to achieve, it is time to apply it to a real world example using two

different mechanisms, a variation of the initial example, the randomized response mechanism, and the Laplace mechanism.

2.2.1 The Randomized Response Mechanism

The example used to illustrate how Differential Privacy works can be extended as shown in Algorithm 1.

Algorithm 1: $M_R(d, \alpha, \beta)$:

1. Flip a biased coin with probability of heads α;
2. If heads, then answer truthfully with d;
3. If tails, flip a coin with probability of heads β and answer "yes" for heads and "no" for tails.

To estimate the proportion p of participants whose answer is "yes" we have to do some changes to our previous estimation method, given that now the coin is biased. We know that:

$$P(X_i = 1) = \alpha p + (1 - \alpha)\beta$$

Solving for p we get:

$$p = \frac{P(X_i = 1) - (1 - \alpha)\beta}{\alpha}$$

Given a sample of size n , we can estimate $P(X_i = 1)$ with $\frac{\sum_{i=1}^{i=n} X_i}{n}$. Then, the estimate $\hat{p}$ of p is:

$$\hat{p} = \frac{\frac{\sum_{i=1}^{i=n} X_i}{n} - (1 - \alpha)\beta}{\alpha}$$

Now that we can estimate the proportion of positive answers for a population, it will be interesting to know the accuracy of our estimation. For this we can use the variance assuming that each response is independent:

$$\mathrm{Var}(\hat{p}) = \mathrm{Var}\left(\frac{\sum_{i=1}^{i=n} X_i}{n\alpha}\right) = \frac{\mathrm{Var}(X_i)}{n\alpha^2}$$

If we take the square root of the variance it is possible to determine the standard deviation s of $\hat{p}$, which is proportional to $\frac{1}{\sqrt{n}}$, since the other factors are not dependent on the number of participants. Multiplying both $\hat{p}$ and s by n yields the estimate of the number of participants that answered "yes" and its relative accuracy expressed in number of participants, which is proportional to $\sqrt{n}$.

We can also calculate the level of differential privacy. Let's imagine we have a dataset d with two configurations, d_yes, containing the positive answers each one represented by a 1, and d_no, which contains the negative answers represented by

a 0. We know that $||d_i - d_j||_1 \leq 1$ for any i, j, as defined by Differential Privacy. If we apply that definition to our mechanism we get:

$$\frac{Pr[M_R(d_i, \alpha, \beta) = k]}{Pr[M_R(d_j, \alpha, \beta) = k]} \leq e^{\epsilon}$$

Replacing i and j by our configurations, we can calculate the level of differential privacy for our mechanism d:

$$\frac{Pr[M_R(d_{yes}, \alpha, \beta) = 1]}{Pr[M_R(d_{no}, \alpha, \beta) = 1]} \leq e^{\epsilon}$$
$$\ln\left(\frac{\alpha + (1-\alpha)\beta}{1 - (\alpha + (1-\alpha)\beta)}\right) \leq \epsilon$$
$$\ln\left(\frac{\alpha + \beta - \alpha\beta}{1 - (\alpha + \beta - \alpha\beta)}\right) \leq \epsilon$$

The level of differential privacy is represented by ϵ and can be tuned varying the parameters α and β. For a level of $\ln 3$ we can set $\alpha = \frac{1}{2}$ and $\beta = \frac{1}{2}$ (this is, the same probability of getting head or tail, which is the original example using an unbiased coin).

2.2.2 The Laplace Mechanism

The Laplace mechanism adds Laplace noise from the Laplace distribution centered at 0 with scale $b = \frac{1}{\epsilon}$

$$Lap(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}$$

So it can be defined as:

$$M_L(x, f, \epsilon) = f(x) + Y$$

where f is a function and Y is a random variable drawn from

$$Lap(b) = Lap(\frac{1}{\epsilon})$$

Following the same procedure used with the Randomized Response mechanism, we need to prove that the Laplacian Mechanism preserves ϵ Differential Privacy.

Given two datasets x, y such that $||x - y||_1 \leq 1$, as stated by the definition of Differential Privacy, and a function f which returns a real number from a dataset, let p_x denote the probability density function of $M_L(x, f, \epsilon)$ and p_y the probability density function of $M_L(y, f, \epsilon)$. We can compare the two of them at some arbitrary point z:

$$\frac{p_x(z)}{p_y(z)} = \frac{e^{-\epsilon|f(x)-z|}}{e^{-\epsilon|f(y)-z|}} = e^{\epsilon(|f(x)-z|-|f(y)-z|)} \leq e^{\epsilon|f(x)-f(y)|}$$

Then,

$$e^{\epsilon|f(x)-f(y)|} \leq e^{\epsilon}$$

It is also possible to compute the accuracy of the Laplace mechanism. As said before, $Y \sim Lap(b)$, then $Pr[|Y| \geq t \times b] = e^{-t}$. Hence, let $k = M_L(x, f, \epsilon)$ and $\forall \delta \in (0, 1]$:

$$Pr\left[|f(x) - k| \geq \ln\left(\frac{1}{\delta}\right) \times \frac{1}{\epsilon}\right] = Pr\left[|Y| \geq \ln\left(\frac{1}{\delta}\right) \times \frac{1}{\epsilon}\right] = \delta$$

The accuracy in the Laplace mechanism, unlike in the randomized response, does not depend on the number of participants n.

2.2.3 Randomized Response and Laplace mechanism: Comparison

So far, we have seen two different mechanisms that meet Differential Privacy. In this section we will see how the same query can be answered by different mechanisms with the same level of Differential Privacy.

Choosing the right Differential Privacy mechanism depends on the problem to solve. Different mechanisms differ in the accuracy of the results, the performance, or whether they require the original dataset or not.

Let's imagine we perform a study on n individuals. In this study we want to know how many of those individuals possess some property p. We can represent each individual with a Bernoulli random variable.

The Randomized Response mechanism would be represented as $M_R(d, \frac{1}{2}, \frac{1}{2})$. It satisfies $\ln 3$ differential privacy, as we have proved previously. On the other hand, the Laplace mechanism would be represented as $M_L(d, f, \ln 3)$, which also satisfies $\ln 3$ differential privacy.

Although both mechanisms satisfies the same level of differential privacy, M_R is applied to each individual response and combined later into a single result, while M_L is applied directly to the entire dataset. This means that if the Laplace Mechanism is used, the original data should be collected first, and then differential privacy is applied to the aggregated data, in such a way that the possibility that an attacker might get access to it exists.

In contrast, the Randomized Response mechanism applies the noise directly to the individual responses of the users and only the noisy responses are collected. With this mechanism, any individual's information cannot be learned with certainty, but an aggregator can still infer population statistics.

But as everything in Computer Science, this is a matter of tradeoffs. Although both mechanisms satisfy the same level of differential privacy, M_L needs the original data, but the accuracy of the results is also higher.

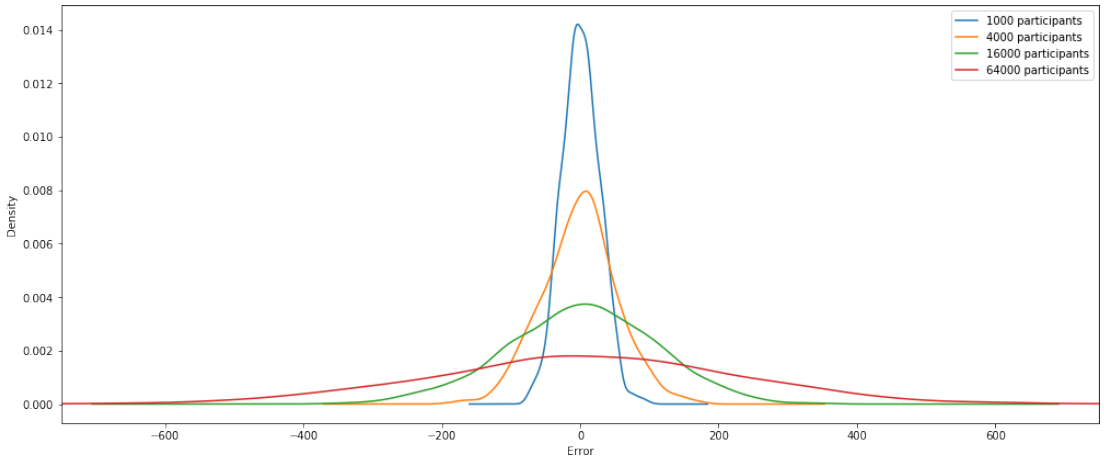

Figure 1: Accuracy using the Randomized Response mechanism. [13]

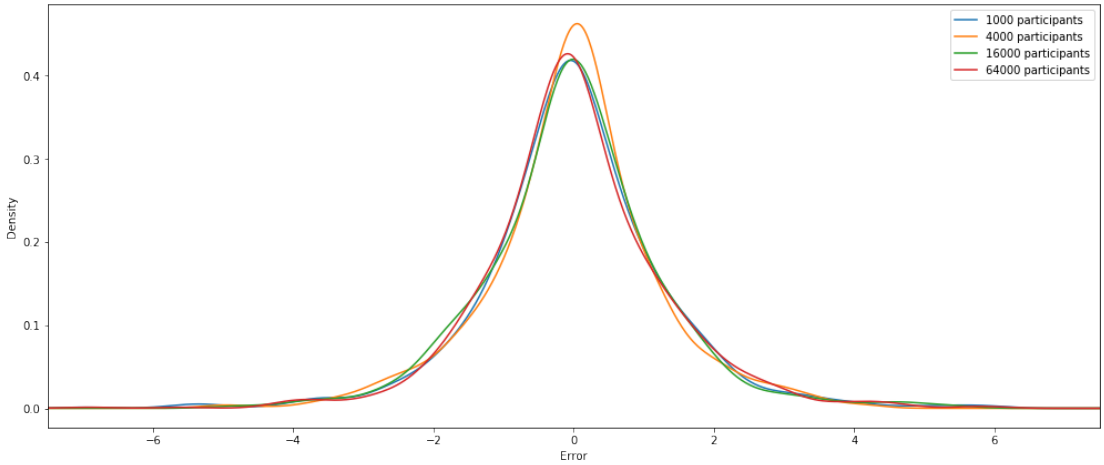

Figure 2: Accuracy using the Laplace mechanism. [13]

Which mechanism to use depends, as it has been said, on several factors. In the medical field, for example, one may trust data collectors but not the community who will be accessing the data. The Laplacian mechanism would be appropriate in this case. Nonetheless, in the online world, users try to protect their privacy and do not trust in the companies that collect their data, so the Randomized Response mechanism would be preferred even if it provides less accuracy.

2.3 RAPPOR

RAPPOR [10] stands for Randomized Aggregatable Privacy-Preserving Ordinal Response. It is an algorithm developed at Google to collect data while preserving the privacy of its users adding random noise to guarantee Differential Privacy.

It is based on the Randomized Response mechanism explained in 2.2.1

The algorithm is divided in two parts: the data collection and anonymization, and the decoding and analysis.

The parameters used are:

- **Size of the Bloom filter, k:** RAPPOR uses a Bloom filter to report the data. When selecting the bloom filter size we should have in mind how many unique values are expected.
- **Number of hash functions, h:** Bloom filters use hash functions, that map data of arbitrary size to data of fixed size. In this case from a string of characters to an index in the Bloom filter, to encode the values.
- **Number of cohorts, m:** To avoid collisions, RAPPOR divides the population into different cohorts. This value must be chosen carefully. If it is too small, collisions are still quite likely, while if it is too large then each individual cohort provides insufficient signal due to its small sample size.
- **Probabilities p, q, f:** Noise is added to the Bloom filter with different probabilities. These probabilities determine the level of Differential Privacy along with the number of hash functions used.

2.3.1 Data collection and anonymization

The data collection and anonymization consists of four steps that are further explained: 1. Collect and encode the original data. 2. Permanent Randomized Response. 3. Instantaneous Randomized Response. 4. Send the encoded data.

Collect and encode the original data
RAPPOR encodes strings of characters into a Bloom Filter. This Bloom filter never leaves the client.

What is a Bloom Filter
RAPPOR encodes the values into a Bloom Filter. But, what is a Bloom Filter, and why is it used?
A Bloom Filter is a probabilistic data structure used to test whether an element belongs to a set. False positive matches are possible, but false negatives are not. It is represented as a bit array of m bits, all set to 0. There must be also k hash functions that map or hash some set of elements to one of the m array positions. To add an element, hash it using the k hash functions to get k positions in the array. Afterwards, set the bits corresponding to that positions to 1.
To check if an element is in the Bloom filter, hash it using the k hash functions to get k positions in the array. Check if all those positions contain the bit set to 1. If they are, that element may be present in the Bloom filter. It can be the case that the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

To do this, h hash functions are used to encode the value into the Bloom Filter B of size k.

In order to use small Bloom filters, the users are assigned to a cohort randomly, and each cohort implements a different set of h hash functions.

Permanent Randomized Response
The bits of the Bloom filter are set to 0 or 1 with probability $f/2$, or remains unchanged with probability $1 - f$. The resulting Bloom filter should be stored in the client and used in the future if the client needs to report the same value more than once.

In other words, for each client's value v and bit i, $0 \leq i < k$ in B, create a binary reporting value B'_i which equals to:

$$B'_i = \begin{cases} 1, \text{ with probability } \frac{1}{2}f \\ 0, \text{ with probability } \frac{1}{2}f \\ B_i, \text{ with probability } 1 - f \end{cases}$$

Instantaneous Randomized Response
The Instantaneous Randomized Response is computed using the probabilities p and q. The resulting Bloom filter will have the bit in position i set to 1 with probability q if its value was 1 in the Permanent Randomized Response, or with probability p if its value was 0.

$$P(S_i = 1) = \begin{cases} q, \text{ if } B'_i = 1 \\ p, \text{ if } B'_i = 0 \end{cases}$$

Send the encoded data
The resulting data from the previous step is sent for analysis. In Figure 3 we can

see an example of how the encoding mechanism of RAPPOR works.

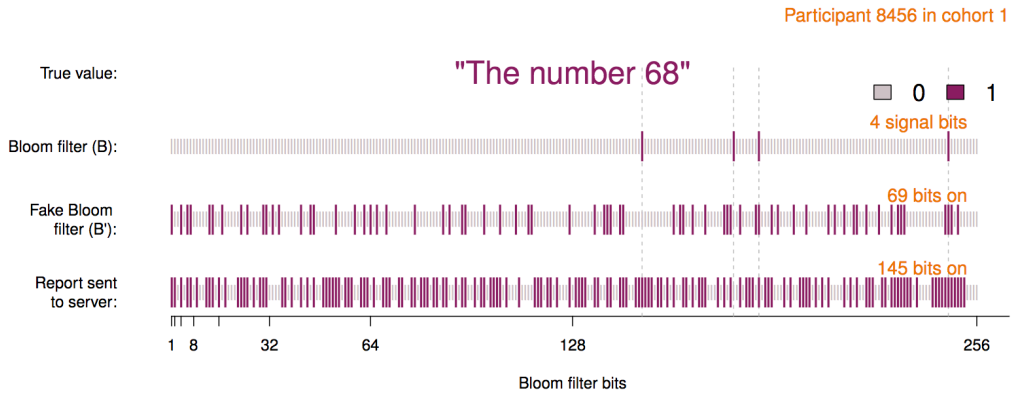

Figure 3: Example of how RAPPOR encodes a value. The client value of the string "The number 68" is hashed onto the Bloom filter B using h (here 4) hash functions. For this string, a Permanent randomized response B' is produced and memoized by the client, and this B' is used (and reused in the future) to generate Instantaneous randomized responses S (the bottom row), which are sent to the collecting service. Source: RAPPOR, Google, 2014.

2.3.2 Data decoding and analysis

After clients have generated their randomized responses, they send them to a server. This server has the task of aggregating the reports and figuring out which answers were actually given, and how often. To do this, we make use of statistical techniques.

The bit arrays are the only information we get from the clients. However, because of the Bloom filter, there are generally infinitely many answers that lead to the same bits being set. This means that we need some set of answers that we explicitly check for. We call this the candidate set. What values are used as candidates is completely dependent on the data that we are collecting.

Of course we know what bits would be set when hashing these candidate values. If we would also know how often each bit was truly set in the original Bloom filters, before noise was added, then we could model this problem using an equation system. In this system, we are looking for candidate counts so that the bits set equal the true number of times the individual bits were set. In statistics, this corresponds to a regression problem.

Estimating the counts of bits

Of course, the whole point of differential privacy is that we do not have access to the original set of bits, so we cannot directly solve this hypothetical equation system. But what we can do is to figure out estimates for how often the bits were set. This is possible because we can estimate how much noise was added

on average. Going into detail for the exact formulas do not help much to build intuition, and they can be found in [10].

While this approach of making estimates might sound a bit messy, it actually has a fairly good theoretical backing. By the law of large numbers [1], the estimate will converge to the true counts with an increasing amount of data. This also explains one very important constraint when using differential privacy: We need a lot of users to make sense of the data. In order to make the estimates more accurate, we will need more users. On the other hand, we cannot control for the random noise well enough if we do not have a good amount of data.

After having estimated how often bits were changed for the randomized response, we can compute estimates for how often the bits were set in the original Bloom filter. We'll call these estimates our target vector y. Note that this only gives us information about how often bits were set in total, across all users. We have absolutely no clue about which users had the bits set in their original Bloom filter.

Example

To give a more concrete idea of where we are going with this, we can stop for a moment and consider this simple example. Let's say we use a Bloom filter with three bits and two hash functions. After having received the randomized reports from enough users, we estimate the following true bit counts:

- bit 1: 3000
- bit 2: 4000
- bit 3: 1000

We are collecting data where each client can give exactly one answer out of the possible answers a, b and c. These values correspond to our candidate set. When hashing the candidate values, the following bits would be set:

- a: bits 1, 2 would be set
- b: bits 1, 3 would be set
- c: bits 2, 3 would be set

Given this information, we are looking for counts of how often the answers a, b or c were given so that we arrive at the estimated numbers for the individual bits. The important inside here is that this is an equation system:

- bit 1: $count_a + count_b = 3000$
- bit 2: $count_a + count_c = 4000$
- bit 3: $count_b + count_c = 1000$

Note how this is not just any kind of equation system, it is a linear equation system. This is great as there are many well-known ways to solve linear equation

[1] https://en.wikipedia.org/wiki/Law_of_large_numbers

systems. The straightforward solution for this specific system is that answer a was given 3000 times, b was never given, while c was given 1000 times.

Of course this is a very simple and artificially constructed example, it is just meant to showcase the problem that the RAPPOR analysis is being reduced to.

Creating the data matrix X

Linear equation systems can generally be well presented using matrices and vectors. We already described out target vector y earlier. What is left to talk about is the data matrix X. This matrix encodes what bits are set when candidate values are hashed in different cohorts.

The general idea here is that for each bit and cohort we add a row to the matrix and for each candidate value we add a column. A cell then has value 1 if the corresponding bit is set when hashing the corresponding candidate value in the respective cohort. Otherwise, it has value 0.

In the above simple example, where we have no cohorts, X would look like this:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Now, our linear equation system can be described by $X * b = y$ where b gives us the candidate counts that explain the set bits.

Linear regression

Usually, we cannot directly solve this equation system. One reason is that y only contains estimates and that our candidate set might be incomplete. This means that the equation system might not have a perfect solution and that we are generally only looking for an approximate one. However, this is still a fairly standard problem in statistics and is usually solved by fitting a linear regression model.

The other problem is that our system does not entirely consist of linear equations. It would not make sense to have negative counts. Thus, b may only contain non-negative values. This makes the problem a fair bit harder to solve, but again it is not a completely new problem. There are some implementations of non-negative least squares solvers available that allow us to find the best approximate solution to a linear equation system with the non-negativity constraint.

Significance tests

It is worth keeping in mind that we are only operating on estimates and that some hash collisions are possible. There are many different approximations for the linear

equation system and it is not clear whether candidates with very small counts were actually reported in the original Bloom filters.

All of this need a statistical significance tests. Computing p-values for linear regression coefficients is a standard practice and is usually done using t-tests. We can use one-sided t-tests because the non-negativity constraint means that extreme results are only possible in one direction.

A significance level of 0.05 to filter out candidate values that do not have enough evidence for their associated counts is used. Because there might be a lot of candidate values, a Bonferroni corrected significance level is used. Finally, only the candidate values and frequencies that we have enough confidence in are reported.

2.3.3 RAPPOR Variants

There are three different variants of this algorithm depending on how many times the data is submitted from a client and the number of unique values:

- **One-time RAPPOR:** The data is sent only once by client. This version does not require longitudinal privacy protection. The Instantaneous randomized response step can be skipped in this case and a direct randomization on the true client's value is sufficient to provide strong privacy protection
- **Basic RAPPOR:** If the set of unique values is small, it is possible to map them to a single bit in the bit array, so it is not necessary to use a Bloom filter with several hash functions.
- **Basic One-time RAPPOR:** This third variant is the combination of the previous ones. One round of randomization using a deterministic mapping of strings into their own unique bits. It is the simplest version of RAPPOR.

2.4 Applications of Differential Privacy

Several companies are applying differential privacy to collect data efficiently. Uber [14] introduced a technique for efficiently calculating the sensitivity of a query without requiring changes to the database. They call it *Elastic Sensitivity* and it applies differential privacy to SQL queries. In contrast with RAPPOR, it does not use local differential privacy, as the original data is still collected and saved in the database.

Apple [15] also uses Differential Privacy to collect usage data in iOS. They use a similar approach to the one shown in RAPPOR, in which they collect data using local differential privacy applying an algorithm called Private Count Mean Sketch algorithm, and this data is sent to a server, where it is analyzed.

Google uses RAPPOR [10] to collect usage data in its browser, Chrome [2]. It was

[2]http://www.chromium.org/developers/design-documents/rappor

the fist commercial deployment of differential privacy.

3 Objectives

In this section, the goals that we are trying to achieve with this project are explained along with how we expect to achieve them. First, we will show what we want to achieve with RAPPOR, and later, how we want to implement this algorithm in Firefox and why.

Now that the State of the Art has been exposed, we understand that for most of the situations, removing identifiers from the data is not enough to ensure privacy. We have also seen that there is a novel research field called Differential Privacy that claims to solve the stated problems.

Most of the applications and research in Differential Privacy have been done with academic purposes in mind, but the industry is starting to develop Differential private mechanisms to collect data. One of this mechanisms is RAPPOR.

This section is divided in two parts, each one explaining a different objective of this thesis. First, we will study how differential privacy and RAPPOR work, and later, we will implement a solution using this technique.

3.1 Working with Differential Privacy through RAPPOR

Unfortunately, the real world is different to the academic world, and research that shows promising results in a controlled environment can behave differently when applied in a real world situation.

The first objective of this project is to simulate RAPPOR and demonstrate that it can be applied to collect data from real users.

The goal of this simulations is:

- **Determine the correct parameters:** As we have explained in Section 2.3, RAPPOR has several parameters that need to be tuned. These parameters affect the performance and accuracy of the results. The correct parameters can make a noticeable difference.
- **Determine the minimum number of clients needed:** One of the main purposes of the simulations is to determine how many clients are needed to get useful results. This is important because the data we are collecting contains noise and the underlying distribution can only be inferred if we have enough data.
- **Determine the expected accuracy of the data:** Given a certain number of clients and an approximation of the unique values expected, what is the accuracy we should expect? This is an important question, as we need to know how much we can trust in the results obtained.
- **Determine the best level of Differential Privacy:** The correct value for the level of Differential Privacy is still an open question. Nobody really knows

what is the best value to guarantee privacy and have a good performance. We will then assume that the value of this parameter depends on the problem to solve, and we will try to come up with the best value for our problem.

3.2 Build a SHIELD Study for Firefox

The second objective is to implement RAPPOR in Firefox. Firefox is a web browser used by millions of people every day. One of their premises is that they care about the privacy of their users. Nonetheless, they also need data to improve their products. They already collect anonymous data [3], but they cannot collect data that could potentially identify a user.

Using Differential Privacy and RAPPOR, we would be able to collect more sensitive data, such as top sites users visit and how features perform on specific sites, while ensuring privacy, in a unbiased way, as the user would not need to give explicit consent.

The add-on developed will collect effective top-level domain and the domain name (eTLD+1), e.g. facebook.com or google.co.uk. with the idea of answering these questions:

- Which top sites are users visiting? – If the most visited sites are known, Firefox can be optimized to work better in those sites.
- Which sites using Flash does a user encounter? – Knowing how frequent a user finds a site using deprecated technologies such as Flash can help Mozilla to take decisions.

3.3 Use Cases

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements.

ID	UC-XX
Title	
Actor	
Preconditions	
Description	

Table 5: Use Cases template

All the use cases that will be covered will be specified in individual tables following the template in the Table 5. Each of the use cases will receive a unique identifier of the format `UC-XX`, where `XX` is a double digit number. This identifier will be used later on different matrices to trace requirements.

[3] https://wiki.mozilla.org/Firefox/Data_Collection

ID	UC-01
Title	**Simulate RAPPOR**
Actor	User
Preconditions	The user has the RAPPOR simulator installed and all the required dependencies.
Description	The user executes the RAPPOR simulator with the parameters (distribution, number of clients, number of unique values, values per client, probabilities and cohorts) they want.

Table 6: Use Case UC-01. Simulate RAPPOR

ID	UC-02
Title	**Simulate RAPPOR and get statistics**
Actor	User
Preconditions	The user has the RAPPOR simulator installed and all the required dependencies.
Description	The user executes the RAPPOR simulator with the parameters (distribution, number of clients, number of unique values, values per client, probabilities and cohorts) they want. Useful statistics are shown at the end of the simulations (total variation distance, false positive rate, false negative rate, number of detected values).

Table 7: Use Case UC-02. Simulate RAPPOR and get statistics

ID	UC-03
Title	**Run several RAPPOR simulations and get statistics**
Actor	User
Preconditions	The user has the RAPPOR simulator installed and all the required dependencies.
Description	The user executes the RAPPOR simulator with the parameters (distribution, number of clients, number of unique values, values per client, probabilities and cohorts) they want for several simulations. Useful statistics are shown at the end of the simulations (total variation distance, false positive rate, false negative rate, number of detected values) comparing the different simulations.

Table 8: Use Case UC-03. Run several RAPPOR simulations and get statistics

ID	UC-04
Title	**Send user's homepage eTLD+1 encoded with RAPPOR**
Actor	Browser
Preconditions	The user has Firefox installed, it allows Mozilla to run studies, and the RAPPOR study is installed.
Description	The add-on gets the user's homepage, extracts the eTLD+1 and encodes the value with RAPPOR.

Table 9: Use Case UC-04. Send user's homepage eTLD+1 encoded with RAPPOR

ID	UC-05
Title	**Simulate RAPPOR in Firefox**
Actor	User
Preconditions	The user has Firefox installed, it allows Mozilla to run studies, and the RAPPOR study is installed.
Description	A simulation is run in the add-on inside Firefox with generated values to test that the algorithm is working as expected.

Table 10: Use Case UC-05. Simulate RAPPOR in Firefox

3.4 Requirements

Now that the Use Cases have been defined, we can define the requirements, both functional and non-functional. A functional requirement is any requirement which specifies what the system should do. In contrast, a non-functional requirement specifies how the system performs a certain function.

Every functional requirement, that specifies a function that has to be offered by the simulator and the add-on, will be tagged using an unique identifier with the format FR-XX, where the XX represent a double digit number.

ID	FR-XX || NFR-XX
Title	
Description	
Priority	
Use-case(s)	

Table 11: Requirements template

Every non-functional requirements, associated with preconditions or other context necessary for the add-on and the simulator to work correctly, will be identified with the tags NFR-XX, where the XX stands for a double digit number.

3.4.1 Functional Requirements

ID	FR-01
Title	**Use MD5 as hash function**
Description	The original values have to be encoded into a Bloom filter using a hash function. MD5 is a well-known hashing algorithm.
Priority	High
Use-case(s)	UC-01, UC-02, UC-03, UC-04, UC-05

Table 12: Functional Requirement FR-01. Use MD5 as hash function

ID	FR-02
Title	**Use SHA256 as hash function**
Description	The original values has to be encoded into a Bloom filter using a hash function. SHA256 is a well-known hashing algorithm.
Priority	High
Use-case(s)	UC-01, UC-02, UC-03, UC-04, UC-05

Table 13: Functional Requirement FR-02. Use SHA256 as hash function

ID	FR-03
Title	**Encode correctly the eTLD+1 of the homepage**
Description	The algorithm should encode correctly in a bit string the value of the eTLD+1 of the user's homepage in Firefox.
Priority	High
Use-case(s)	UC-04, UC-05

Table 14: Functional Requirement FR-03. Encode correctly the eTLD+1 of the homepage

ID	FR-04
Title	**Send a Telemetry ping with the encoded value**
Description	The Firefox extension sends a Telemetry ping with the encoded information.
Priority	High
Use-case(s)	UC-04, UC-05

Table 15: Functional Requirement FR-04. Send a Telemetry ping with the encoded value

ID	FR-05
Title	**Generate plots showing the original and the detected distributions**
Description	Generate two plots showing the original and detected distributions to visually see the accuracy of the algorithm.
Priority	High
Use-case(s)	UC-02, UC-03

Table 16: Functional Requirement FR-05. Generate plots showing the original and the detected distributions

ID	FR-06
Title	**Compute the total variation distance among two distributions**
Description	One of the metrics to compute should be the total variation distance.
Priority	High
Use-case(s)	UC-02, UC-03

Table 17: Functional Requirement FR-06. Compute the total variation distance among two distributions

ID	FR-07
Title	**Compute the number of detected values using RAPPOR**
Description	Offer the number of detected values to determine how accurate is the algorithm.
Priority	High
Use-case(s)	UC-02, UC-03

Table 18: Functional Requirement FR-07. Compute the number of detected values using RAPPOR

ID	FR-08
Title	**Compute the false positive rate**
Description	Show the false positive rate of the detected values.
Priority	Medium
Use-case(s)	UC-02, UC-03

Table 19: Functional Requirement FR-08. Compute the false positive rate

ID	FR-09
Title	**Compute the false negative rate**
Description	Show the false negative rate of the detected values.
Priority	Medium
Use-case(s)	UC-02, UC-03

Table 20: Functional Requirement FR-09. Compute the false negative rate

ID	FR-10
Title	**Generate a plot showing the total variation distance between several simulations**
Description	Show the total variation distance among several simulations to see how the accuracy varies.
Priority	Medium
Use-case(s)	UC-02, UC-03

Table 21: Functional Requirement FR-10. Generate a plot showing the total variation distance between several simulations

ID	FR-11
Title	**Generate a plot showing the false positive rate among several simulations**
Description	Show the false positive rate among several simulations to see how the accuracy varies.
Priority	Medium
Use-case(s)	UC-02, UC-03

Table 22: Functional Requirement FR-11. Generate a plot showing the false positive rate among several simulations

ID	FR-12
Title	**Generate a plot showing the false negative rate among several simulations**
Description	Show the false negative rate among several simulations to see how the accuracy varies.
Priority	Medium
Use-case(s)	UC-02, UC-03

Table 23: Functional Requirement FR-12. Generate a plot showing the false negative rate among several simulations

3.4.2 Non-functional Requirements

ID	NFR-01
Title	**Operating System for the simulator**
Description	The simulator can be used out of the box in macOS.
Priority	High

Table 24: Non-Functional Requirement `NFR-01`. Operating System for the simulator

ID	NFR-02
Title	**Operating System for the study**
Description	The study can run in any operating system that supports Firefox.
Priority	High

Table 25: Non-Functional Requirement `NFR-02`. Operating System for the study

ID	NFR-03
Title	**Firefox requirements**
Description	The study can run only if the user allows Firefox to run studies.
Priority	High

Table 26: Non-Functional Requirement `NFR-03`. Firefox requirements

ID	NFR-04
Title	**Firefox version**
Description	The study has been tested in Firefox 57+.
Priority	High

Table 27: Non-Functional Requirement `NFR-04`. Firefox version

ID	NFR-05
Title	**Python version**
Description	The simulator requires Python 2.7 installed.
Priority	High

Table 28: Non-Functional Requirement `NFR-05`. Python version

ID	NFR-06
Title	**R version**
Description	The simulator requires R 3.4.3 installed.
Priority	High

Table 29: Non-Functional Requirement `NFR-06`. R version

3.4.3 Use Cases to Requirements Traceability Matrix

For a better understanding of the relationship between use cases and requirements, a visual representation in form of traceability matrix is provided. This matrix shows the functional requirements in the rows and the use cases in the columns, to determine which requirements are supposed to satisfy which use cases. The non-functional requirements are not shown in this matrix, as they are stated as preconditions for the system to work as expected.

In Table 30 it is possible to see that all the requirements match two or more use cases, which means that the formulation of the use cases is correct.

	UC-1	UC-2	UC-3	UC-4	UC-5
FR-01	✓	✓	✓	✓	✓
FR-02	✓	✓	✓	✓	✓
FR-03				✓	✓
FR-04				✓	✓
FR-05		✓	✓		
FR-06		✓	✓		
FR-07		✓	✓		
FR-08		✓	✓		
FR-09		✓	✓		
FR-10		✓	✓		
FR-11		✓	✓		
FR-12		✓	✓		

Table 30: Requirement - Module traceability matrix

4 Simulations

As discussed before, before the implementation, a set of simulations must be performed.

This section contains the simulations performed to ensure that the algorithm works as expected, to know how each parameter affects the performance and accuracy of the algorithm, and to select the appropriate parameters for collecting the data in a real world application.

The simulations will be divided in Purpose of the simulation, in where the motivation of the simulation will be explained, Expected result, where the results that we expect to obtain are discussed, and Obtained result, where the obtained results are analyzed.

The simulator is based on the one Google built to test RAPPOR [16], although multiple changes have been introduced, such as support for macOS, and parallel execution to speed up the simulations.

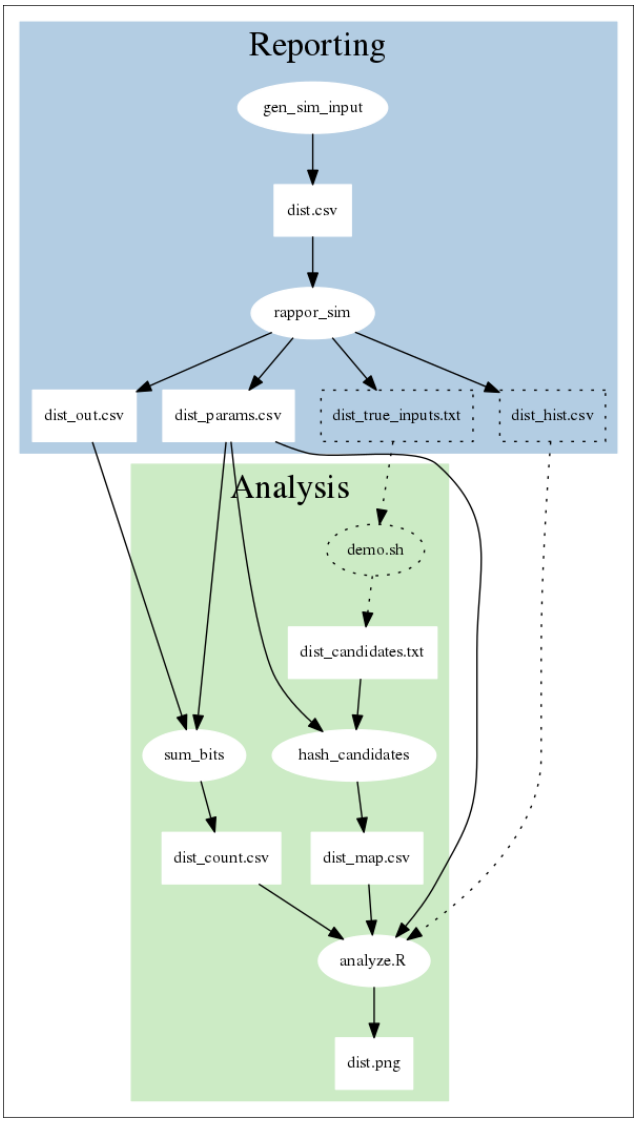

Figure 4: Simulator architecture. [16]

In this simulator, the data is first generated, and then the simulation is executed, creating several files containing the output of the simulation and information about the parameters. In a second step, the data is analyzed and an image comparing the original and the RAPPOR distributions is created.

As it has been explained in Section 2.3, there are several parameters we need to tune to get the optimal results. These parameter are:

- Size of the bloom filter, k.
- Number of clients.
- Number of unique values.
- Values per client.
- Number of hash functions, h.

- Number of cohorts, m.
- Probabilities p, q, f.

4.1 Input and Output

The first step to perform in the simulations is the data generation. The input data for RAPPOR is:

- Value to encode: The value we want to submit in a differential private manner. For the simulations, these values will be randomly assigned to each client following a selected distribution.
- Cohort: Cohort the user belongs to. The cohorts should be randomly assigned and uniformly distributed across the population.

Alongside these two values, our generated data will also contain an identifier of the client that has submitted the data for debugging purposes. In the real implementation, no data capable of identifying a user should leave the client.

```
client,cohort,value
c1,0,v19
c2,1,v54
c3,2,v56
c4,3,v56
c5,4,v79
c6,5,v28
c7,6,v62
c8,7,v46
c9,8,v51
```

Listing 1: Example of generated data used as input.

The output of the simulations consists of the client, the cohort, the original bloom filter, the permanent randomized response and the instantaneous randomized response. The client, original bloom filter and permanent randomized response are written in the output for debugging. In a real implementation, these values should remain private.

```
client,cohort,bloom,prr,irr
c1,1,10100000,10100000,10001000
c2,2,00001100,00001100,00101000
c3,3,01000010,01000010,00001000
c4,4,10000010,10000010,10011010
c5,5,11000000,11000000,01000010
c6,6,10000001,10000001,10011010
c7,7,00001001,00001001,10001011
c8,8,01000100,01000100,11010000
c9,9,00001001,00001001,01101100
```

Listing 2: Example of the output data.

4.2 Evaluation of the results

To evaluate the results obtained in the simulations, first it is necessary to define a metric.

The results will be evaluated based on the values detected by RAPPOR, the false negative rate, and the Total Variation (TV) distance. The total variation distance measures the difference between the ground truth and the values detected by RAPPOR. In other words, it's the L1 distance between the actual and RAPPOR distributions.

4.2.1 Size of the bloom filter

- **Purpose of the simulation:** Determine how the size of the bloom filter affects the accuracy of the results.
- **Expected result:** According to the RAPPOR paper [10], *different sizes perform similarly depending on the values of h and m.* Then, the expected result is that with different values of h (number of hash functions) and m (number of cohorts), different sizes of bloom filters will return uncorrelated results, making it not possible to determine the optimal size of the bloom filter.
- **Obtained result:** The number of strings detected by RAPPOR seems to increase when the size of the bloom filter increases until it reaches 32 bits, as it can be seen in the case of `sim_bloom_filter_1_gauss_small` in Figure 5, where the number of detected strings grows from 59 out of 100 (4 bits used) to 78 out of 100 (256 bits used) and therefore, the Total Variation distance decreases. At this point, the accuracy seems to converge (when the size of the bloom filter is small, the collisions are quite likely, but when this size increases, the collisions decreases until it converges, this is, not improve anymore), or even worsen (in `sim_bloom_filter_4_gauss_small`) as the size of the bloom filter increases. Also, the time it takes to execute RAPPOR increases with the size of the bloom filter. This simulation has been done with 100 unique values and number of hash function (h) of 2 and 4. Increasing the number of hash functions, consumes privacy budget (eventually decreasing ϵ), this loss in privacy budget can be offset by increasing client-side noisification. However, the additional cost of this operation is a decrease in fidelity of the obfuscated data relative to the true underlying data. This allows us to decrease the size of the bloom filter in order to address practical constraints by increasing the number of hash functions.
  - for 1 hash function:

    * for a False Positive Rate of 1%: $k = 100 * n$

  - for 2 hash functions:
    * for a False Positive Rate of 1%: $k = 20 * n$
    * for a False Positive Rate of 0.1%: $k = 60 * n$

  NOTE: n is the number of unique values in a given cohort, not across the entire population. If we estimate the number of unique eTLD+1 values set as homepages in the wild to be 100K. The use of only 1 hash function is intractable.

This plots show the value of the Total Variation distance across different simulations.

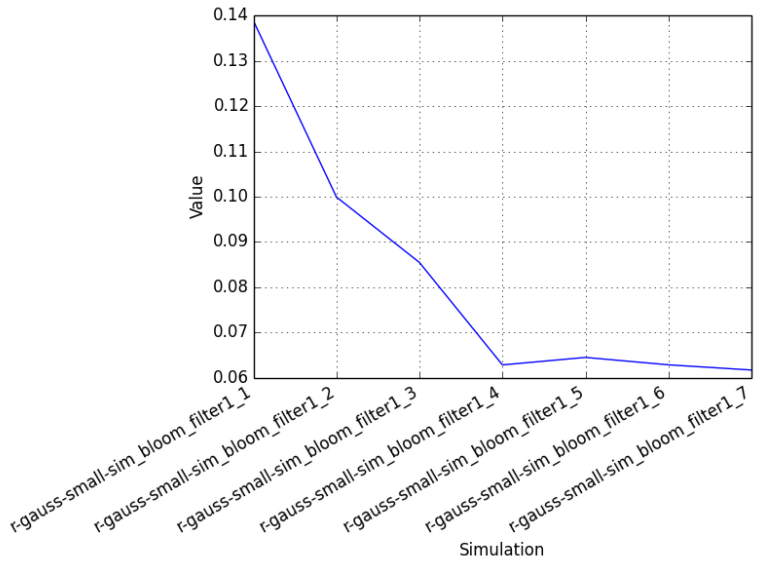

Figure 5: Accuracy when size of the bloom filter k increases.

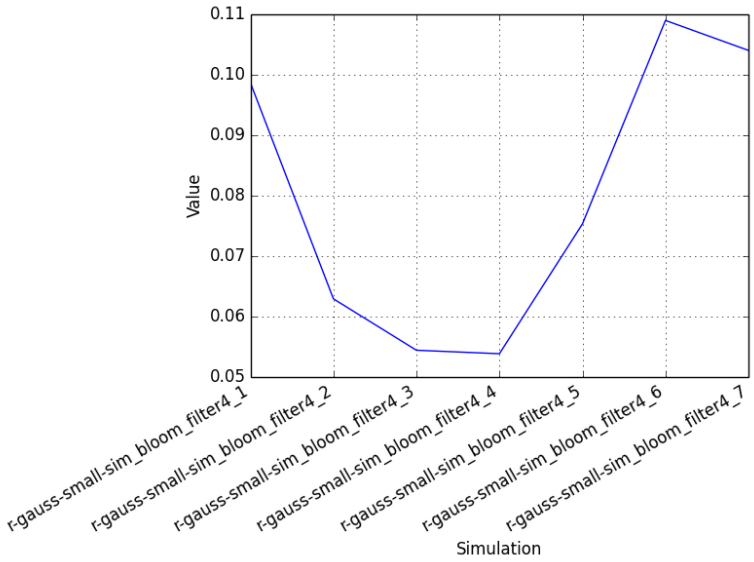

Figure 6: Accuracy when size of the bloom filter k increases.

4.2.2 Number of clients

- **Purpose of the simulation:** How many clients are needed to obtain an accurate result.
- **Expected result:** The larger the number of clients is, the more accurate are the results. We expect that with a small number of clients, the results are not accurate, being the total variation close to 0.5 and with a high false negative rate. These results are expected with all distribution types.
- **Obtained result:** The larger the number of clients (and data submitted) is, the lower is the total variation between the original data and the RAPPOR data. Also, the number of false negatives increases when the number of clients (and total data submitted) decreases. Two different distributions have been tested, *gauss* and *exp*. With 25K clients (and as each client reports 1 value, 25K values) the false negative rate is around 80%, with a total variation distance that lies in between 0.3 and 0.5. When we increase the number of clients, we get that in the case of the Gaussian distribution, with 500K values reported, a false negative rate of 38% is obtained, along with a TV distance of 0.1. In contrast, using the exponential distribution, with the same number of values ($values = clients * valuesperclient$) we get a false negative rate of around 50% but with a TV distance of 0.1. These means that we are getting a good approximation, and although the false negative rate is high (which means that a lot of values has not been detected by RAPPOR), this values are the least important ones. The important values, the ones with higher frequency, are detected and well approximated. With 10M values, in both distributions we get a good approximation of the original data, in which most of the values are detected by RAPPOR (around 80%), and the TV distance lies between 0.02 and 0.03. In conclusion, if

the purpose of RAPPOR is getting an idea of the trending or most used values (ex. eTLD + 1, etc) users have, a population of 500K is enough for 100 unique values. (For a more detailed explanation of how the number of unique values affect the accuracy, see the next simulation). If we want to get a good approximation of the data, the minimum population needed would more than 1M, with 10M reporting accurate results. For now, we can say that the minimum population size needed for a result with a false negative rate between 10% and 20% is $totalValues * 100000$ with $totalValues$ being the number of values reported ($clients * valuesperclient$).

This plots show the value of the Total Variation distance across different simulations.

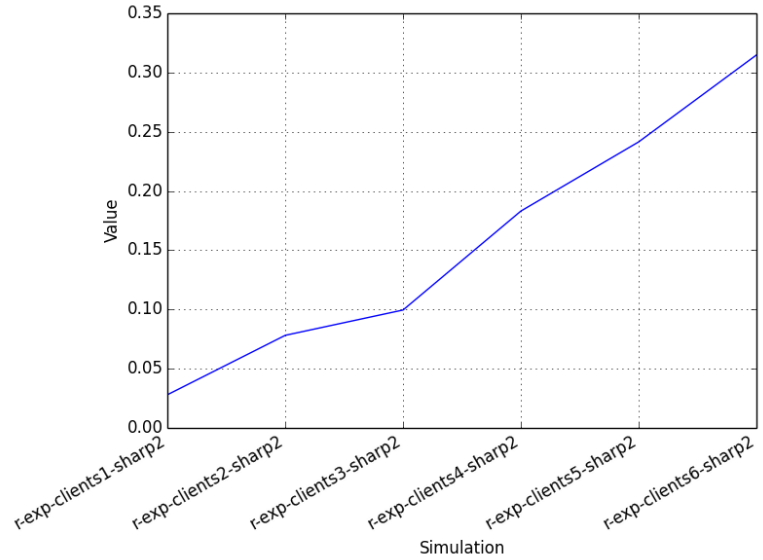

Figure 7: Accuracy when decreasing the number of clients.

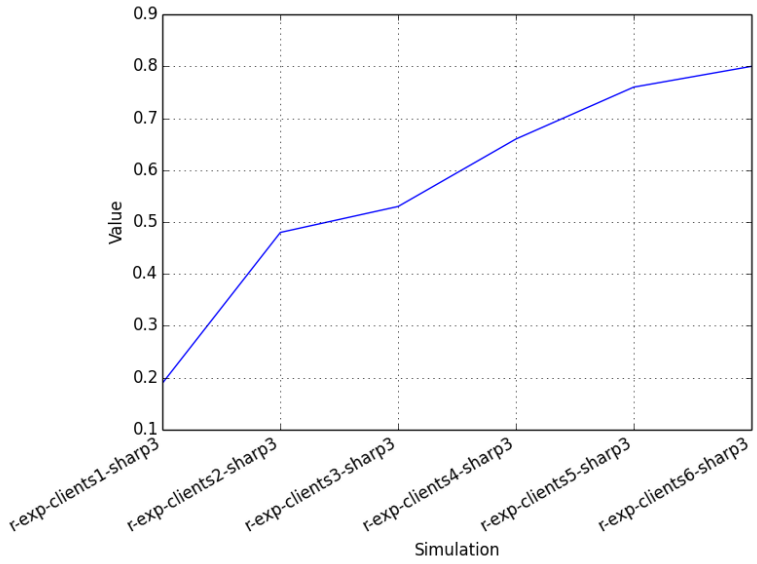

Figure 8: Accuracy when decreasing the number of clients.

4.2.3 Number of unique values

- **Purpose of the simulation:** Maximum number of unique values to get a good result.
- **Expected result:** Given a constant number of clients (and therefore, values reported), the lower the number of unique values is, the more accurate are the results.
- **Obtained result:** As expected, when the number of unique values increases, the accuracy of the results decreases. With 1M clients and 10 unique values, the RAPPOR results are almost perfect. When the number of unique values is increased, the accuracy rapidly decreases. With 250 unique values reported by 1M clients, we can detect the trends but we cannot trust the individual values. With more than 250 unique values (0.00025% of the total number of values reported), the results are invalid. The results are slightly better when an exponential distribution is used. Although the Gaussian distribution returns better Total variance score than the exponential, the trends in the results are easier to identify using the exp. distribution. With 10M clients, the trending values can be detected when having up to 750 values, but if we want accurate results, 250 unique values is the bottom limit. With 500 unique values we get a better approximation than with 750, but once again, we should not trust the frequency of these results and only use them to identify the trending values. With more than 750 unique values (0.000075% of the total number of values reported), the results are invalid. The conclusion of this simulation is that, adding unique values is expensive in terms of clients (or values reported) that are needed to get an accurate approximation.

  In Basic One-Time RAPPOR, a general rule of thumb is that we can reliably detect $sqrt(N)/10$ strings with N being the sample size. This gives us an

upper bound for the number of strings that can be detected in other variants of RAPPOR.

4.2.4 Values per client

- **Purpose of the simulation:** How the number of values reported by each client affects the result.
- **Expected result:** The larger the amount of data is, the more accurate the results are. They should be very similar to the ones obtained in the simulation of the number of clients.
- **Obtained result:** As expected, the results are almost identical to the ones obtained in Number of clients. We can determine then that, it does not matter how many clients are reporting results, but the amount of data reported. (In the case of one-time RAPPOR, each client reports one value, so clients and values in this context are the same) (Note: in this case, each client reports several and different values, not the same value over and over).

4.2.5 Number of hash functions

- **Purpose of the simulation:** How the number of hash functions affects the results.
- **Expected result:** The number of hash functions indicates how many bits in the bloom filter are used to report a value. This is, if the number of hash functions is 2, the bloom filter will have two bits set to 1 in the real reported value (more bits will be set to 1, and some of the original bits will be changed to 0 in the PRR and IRR steps). If the number of hash functions is high, it is very likely that the value of some bits will be changed in the PRR and IRR steps, reporting the original value fewer times.
- **Obtained result:** The result matches the one obtained in the RAPPOR paper (section 4) [10]. The best results are obtained with 2 hash functions, increasing the accuracy from 1 to 2 hash functions, and showing a decrease in this metric when the number of hash functions increases. This has been tested with both exponential and Gaussian distributions, and different number of clients and unique values. The intuition of why this happens is unclear, although it may be due to the probability of a bit being changed in the PRR and IRR steps, as explained in Expected Result. In short, this simulation confirms the results obtained in the RAPPOR paper.

This plot shows the value of the Total Variation distance across different simulations.

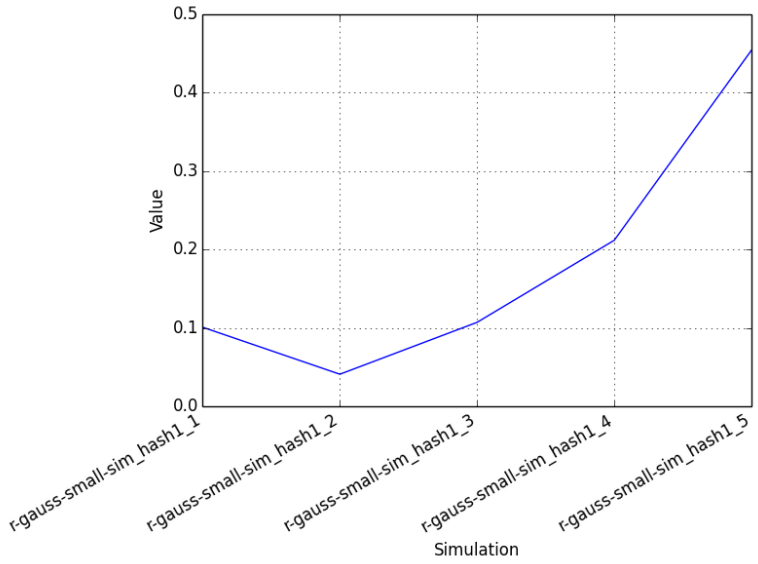

Figure 9: Accuracy when increasing the number of hash functions, `h`.

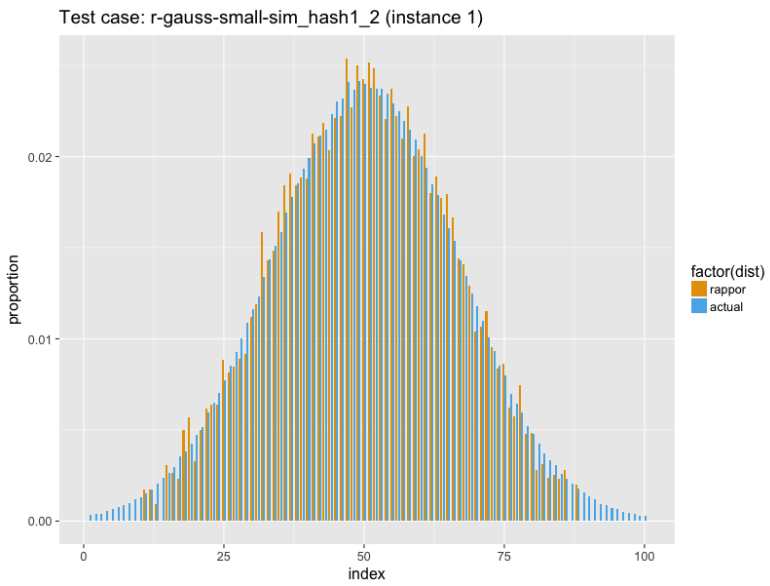

Figure 10: Values detected using 2 hash functions.

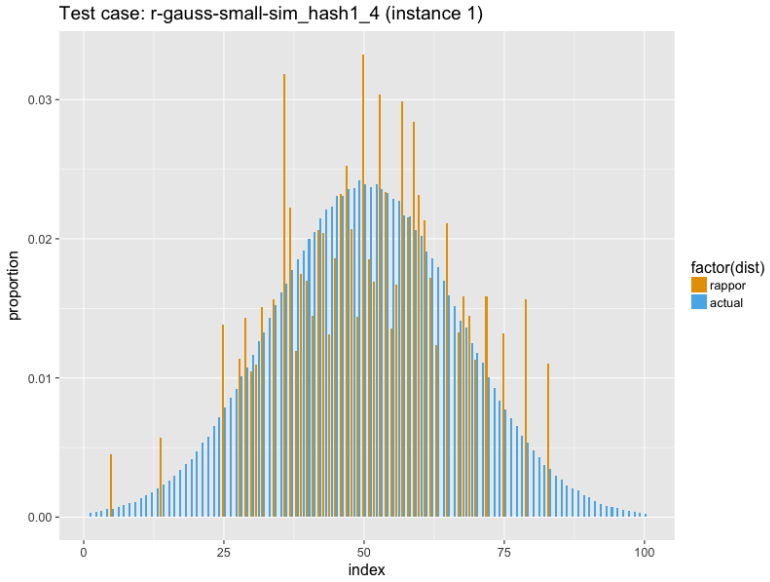

Figure 11: Values detected using 4 hash functions.

4.2.6 Number of cohorts

- **Purpose of the simulation:** How the number of cohorts affects the accuracy of the results.
- **Expected result:** Cohorts implement different sets of h hash functions for their Bloom filters, thereby reducing the chance of accidental collisions of two strings across all of them. The choice of m should be considered carefully, however. When m is too small, collisions are still quite likely, while when m is too large, then each individual cohort provides insufficient signal due to its small sample size (approximately N/m, where N is the number of reports).
- **Obtained result:** Using both Gaussian and exponential distributions with 1M values and 100 unique values, we can see that when the number of cohorts reaches 16, it converges (at least it does not get better or worse testing with up to 256 cohorts). The same results are obtained with 10M clients and 100 unique values. If we increase the population to 10M clients with 1K unique values, the accuracy increases until we reach 128 cohorts. If we double it and we use 256 cohorts, RAPPOR is not able to detect any value. In conclusion, the number of cohorts should be chosen carefully keeping in mind how many clients we are expecting and the number of unique values.

This plot shows the value of the Total Variation distance across different simulations.

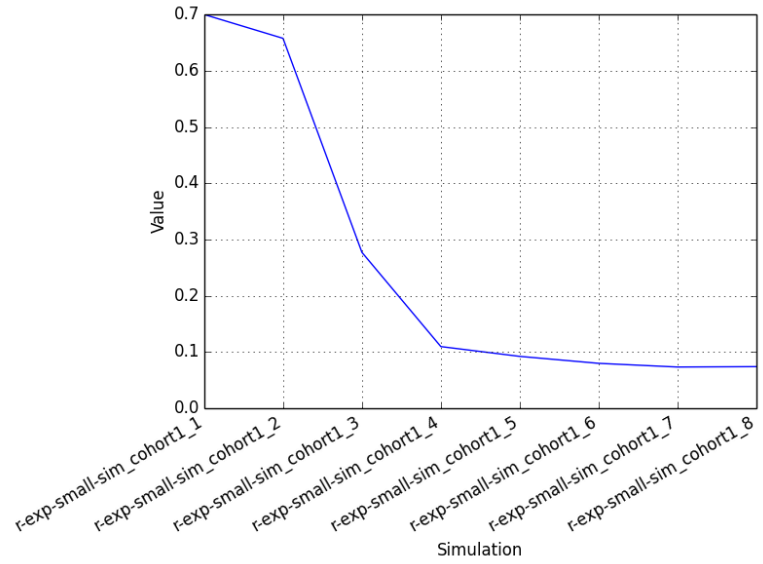

Figure 12: Accuracy when increasing the number of cohorts, `m`.

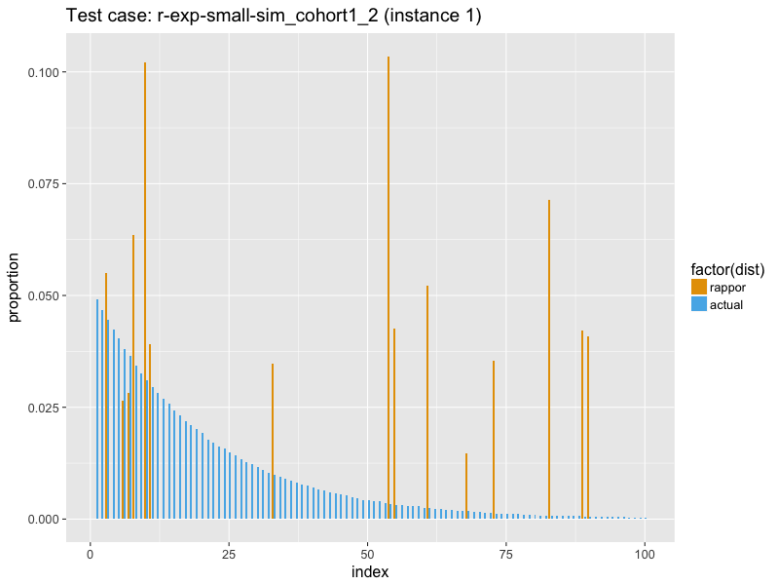

Figure 13: Detected values using 100 unique values, 1000000 clients, 1 value per client, 4 cohorts.

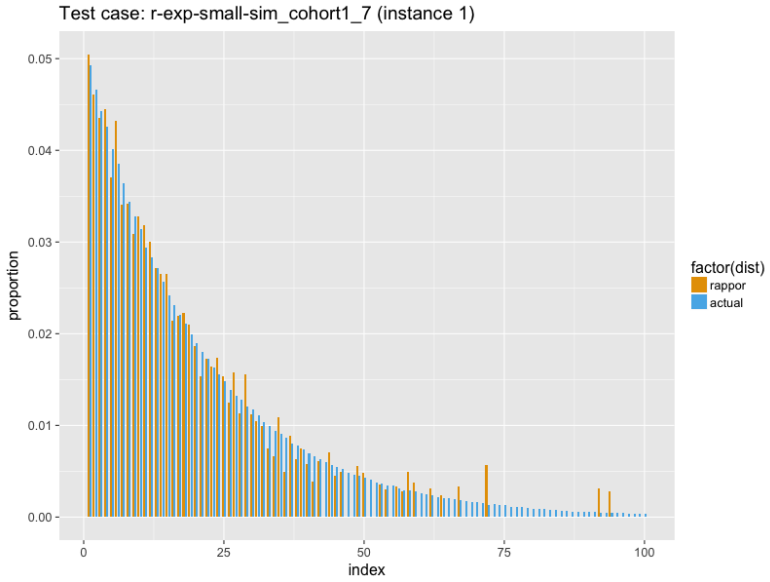

Figure 14: Detected values using 100 unique values, 1000000 clients, 1 value per client, 128 cohorts.

4.2.7 Probabilities

- **Purpose of the simulation:** Testing if, maintaining the same level of DP, changing the parameters p, q and f impacts in the accuracy of the results.
- **Expected result:** The important value is the level of Differential Privacy (in the Appendix you can see how it is calculated), but with the same level of DP, the accuracy of the results should be similar.
- **Obtained result:** As expected, the value that affects directly the accuracy of the results is the level of DP. For this simulation a level of $2 * ln(3)$DP has been used, varying the probabilities p, q and f. With 1M clients and 100 unique values, testing with both gauss and exp distributions, we obtain a Total Variation distance that lies between 0.075 and 0.1, and a false negative rate quite constant. These results indicate that the value for the optimal probabilities p, q and f is unclear, but, as expected, the level of DP should be chosen carefully. On one hand, it has to be strong enough to protect the user, on the other hand, it has to report accurate results that we can analyze.

This plot shows the value of the Total Variation distance across different simulations.

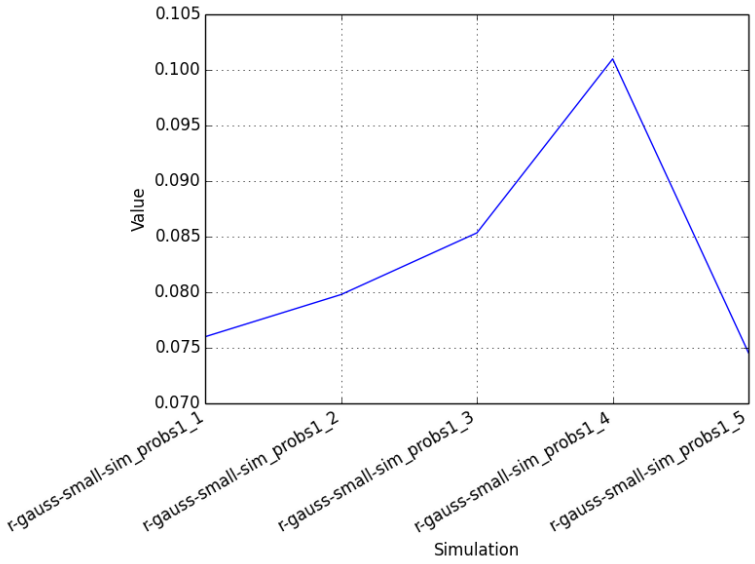

Figure 15: Accuracy when varying the values of probabilities `p, q, f`.

4.3 Conclusions of the Simulations

These simulations have been performed in a controlled environment, with generated data. The main purpose of this is to confirm that the algorithm works as expected. In a real world situation, in which the ground truth is not known, the results can be different.

- **Values needed:** The obvious answer is as much as possible. But a lower bound is needed. This lower bound depends on other parameters, such as the number of cohorts, the number of unique values. The higher is the number of unique values, the higher the number of values reported should be. Depending on the purpose of the simulation, we can have two situations:
  - We want to see the most used values in the data: We do not really care about the frequency of a value in the data, but if these value appears, and how this value is related with other values reported. In this case, as we can see in the second simulation, with 500K values reported and 100 unique values (0.0002%) it can be enough.
  - We want a good approximation of the data: Here, the more values reported, the better. With 10M values reported and 100 unique values (0.00001%), we get a good approximation, as seen in the second simulation.
- If each client only sends the value once, one-time RAPPOR can be used. The results with one-time RAPPOR are supposed to be more accurate, and the complexity of the algorithm is reduced, as the IRR step is skipped.
- The optimum number of hash functions is 2.
- The number of cohorts should be chosen carefully. When the number of

cohorts is too small, then collisions are still quite likely, while when it is too large, then each individual cohort provides insufficient signal due to its small sample size (approximately N/m, where N is the number of reports). it is better to have a high number of cohorts, because as shown in the simulation, when the number of cohorts is too small, the results obtained are useless, but when it increases, the results get better, and in some situations, it converges. We must not forget, however, that a large number of cohorts will result in no detected values.

- The values of the probabilities p, q and f doesn't really matter. What matters is the level of Differential Privacy. It should be chosen taking into account that if the level of DP is strong, the noise in the data could be too high, but if it is too weak , the privacy of the user can be compromised.
- For this project we have decided to choose the parameters p and q in a way that p and q are fully dependent, this is, $q = 1 - p$. Moreover, these values will be chosen solely on the desired value of ϵ (level of DP). For future work, we should study better the relationship between p and q, and how these values can be optimized to create a strong algorithm in terms of privacy guarantee, but that also generates accurate results.

4.4 Parameter selection

The parameters that are going to be used in the Firefox add-on are:

- **Values per client**: 1 – Each client will submit only one value.
- **Cohorts,** m: 100 – According to the expected number of unique values, 100 cohorts should return good results.
- **Bloom filter bits** k: 128 – 128 bit Bloom filter combined with the number of cohorts should be enough to collect the required data.
- **Hash functions,** h: 2 – As shown in the simulations, two hash functions return the best results.
- **Probability** p: 0.35
- **Probability** q: 0.65
- **Probability** f: 0.0
- **Differential Privacy level,** ϵ: The level of Differential Privacy is 2.476. It can be calculated using the values chosen previously.

5 Development

This section will explain the development of the add-on for Firefox implementing RAPPOR and the decisions that were made during the development process.

5.1 Implementation

Extensions add new functionalities to Firefox. They can add anything from a toolbar button to a completely new feature. For this extension, we will not include any visual element, as it is not needed.

Extensions are developed using JavaScript, an interpreted programming language developed for Netscape, the precursor of Firefox. This extension is implemented as a Shield study.

5.1.1 Description of the files

- `HomepageStudy.jsm`: implements the logic to extract the eTLD+1 from the preference and apply RAPPOR.
- `TelemetryRappor.jsm`: implements the RAPPOR algorithm and the related utility functions.
- `StudyUtils.jsm`: miscellaneous Shield utils.
- `bootstrap.js`: contains add-on specific boilerplate code.

5.1.2 Description of architecture

This add-on is structured as a restartless (`bootstrap.js`) extension.

During `bootstrap.js:startup(data, reason)`:

1. `shieldUtils` imports and sets configuration from `Config.jsm`.
2. Modules are imported.
3. Study is set up.
4. RAPPOR is executed.
5. `boostrap.js` waits for requests from `HomepageStudy.jsm` that are study related: `["info", "telemetry", "endStudy"]`.
6. Data is sent to Telemetry.
7. The study ends and the add-on is uninstalled.

5.1.3 Data format

```
{
  "type": "shield-study-addon",
  ... common ping data
  },
  "payload": {
    "version": <int> // 3,
    "study_name": <string> // "TelemetryRAPPOR",
    "branch": <string> // "eTLD+1",
    "addon_version": <string> // "1.0.0",
    "shield_version": <string> // "4.0.0",
    "type": <string> // "shield-study-addon",
    "data": {
      "attributes": {
        "cohort": <string> // "6",
        "report": <string> // "180504828
           f142c0204000004010346c0"
      }
    },
    "testing": false
  },
  ... ping data
}
```

Listing 3: Format of the data sent by the add-on.

- `payload.version`: This field contains the version of the payload.
- `payload.study_name`: This field contains the name of the study, as set in Config.jsm.
- `payload.branch`: This field contains the branch of the experiment. In this case, the experiment has only one branch.
- `payload.addon_version`: This field contains the version of the addon, as set in `package.json`.
- `payload.shield_version`: This field contains the version of the Shield study add-ons library.
- `payload.type`: This field contains the type of the payload. In this case, shield-study-addon.
- `payload.data.attributes.cohort`: This field contains the cohort to which a client belongs to.
- `payload.data.attributes.report`: This field contains the RAPPOR value of the user's homepage.

5.1.4 Shield Study

Shield Studies is a function that prompts a random population of users to help Mozilla try out new products, features and ideas. They are installed as a self-expiring add-on on a certain population (usually, 1% or 2%) of Firefox users. They are a special type of restartless extension, and need to be signed to run in Firefox 42+.

Obtain the data The first step is to get the data. As a reminder, we want to submit the `eTLD + 1` of the homepage. First, we need to obtain the homepage. Once we have this value, it has to be converted to a URI object. Now, it is possible to obtain the value we are looking for. The code for this can be seen in Listing 4.

```
1 const PREF_HOMEPAGE = "browser.startup.homepage";
2 homepage = Services.prefs.getComplexValue(PREF_HOMEPAGE, Ci.
     nsIPrefLocalizedString).data;
3
4 let uriFixup = Cc["@mozilla.org/docshell/urifixup;1"].
     getService(Ci.nsIURIFixup);
5 homepageURI = uriFixup.createFixupURI(homepage, Ci.
     nsIURIFixup.FIXUP_FLAG_NONE);
6
7 eTLD = Services.eTLD.getBaseDomain(homepageURI);
```

Listing 4: Getting the homepage.

This can fail if the host is an IP address or is if it empty when calling `Services.eTLD.getBaseDomain`. In such case, the study ends. For example, if the value stored in the preference is `foo.bar.com` the add-on applies RAPPOR to `bar.com` and then sends the anonymized bit field out. Other possibility is that the user's homepage is `about:home` or other `about:` page. In the case of `about:home`, this is the value we use. In the case of other `about:` page, the reported value is `about:pages`.

Encode the data So far, we have the value we want to submit. This value has to be encoded into a Bloom filter. For this, we have to use a hash function that maps a value to a position in the Bloom filter. Two different encoding methods have been developed using two different hash functions, MD5 and SHA256.

Although is well known that MD5 is not a secure hash function we can use it in this case. The original Bloom filter never leaves the client (in this case, Firefox), and random noise is added, thus an attacker cannot use this fact to decode the data.

In Listing 5 and Listing 6 we can see how to encode the value using MD5 and SHA256.

```
/**
* Hash client's value v (string) onto the Bloom filter B of size k (in bytes) using
* h hash functions and the given cohort.
* @param {string} value - Value to encode.
* @param {integer} filterSize - Size of the bloom filter.
* @param {integer} numHashFunctions - Number of hash functions.
* @param {integer} cohort - Cohort.
*/
function encodeMD5(value, filterSize, numHashFunctions, cohort) {
 let bloomFilter = new Uint8Array(filterSize);
 let hash = Cc["@mozilla.org/security/hash;1"].
    createInstance(Ci.nsICryptoHash);

 // Seed the hash function with the cohort and the hash function number. Since we
 // are using a strong hash function we can get away with using [0..k] as seed
 // instead of using actually different hash functions.
 hash.init(Ci.nsICryptoHash.MD5);
 let seed = String.fromCharCode(cohort);
 let data = bytesFromUTF8(seed + value);
 hash.update(data, data.length);
 let result = hash.finish(false);
 for (let i = 0; i < numHashFunctions; i++) {
   let idx = result.charCodeAt(i);
   // Set the corresponding bit in the bloom filter. Shift 3 bits to select the index, as k is
   // represented in bytes, we need to shift 3 bits to get the corresponding bit (1 byte = 8 bits = 2^3).
   setBit(bloomFilter, idx % (filterSize << 3));
 }
 return bloomFilter;
}
```

Listing 5: Encoding the value using MD5

```
/**
* Hash client's value v (string) onto the Bloom filter B of size k (in bytes) using
* h hash functions and the given cohort.
* @param {string} value - Value to encode.
* @param {integer} filterSize - Size of the bloom filter.
* @param {integer} numHashFunctions - Number of hash functions.
* @param {integer} cohort - Cohort.
*/
```

```
function encodeSHA256(value, filterSize, numHashFunctions, cohort) {
 let bloomFilter = new Uint8Array(filterSize);
 let data = bytesFromUTF8(value);
 let hash = Cc["@mozilla.org/security/hash;1"].createInstance(Ci.nsICryptoHash);
 for (let i = 0; i < numHashFunctions; i++) {
   hash.init(Ci.nsICryptoHash.SHA256);
   // Seed the hash function with the cohort and the hash function number. Since we
   // are using a strong hash function we can get away with using [0..k] as seed
   // instead of using actually different hash functions.
   let seed = bytesFromOctetString(cohort + "" + i);
   hash.update(seed, seed.length);
   hash.update(data, data.length);
   let result = hash.finish(false);
   // The last 2 bytes of the result as the bit index is sufficient for bloom filters
   // of up to 65536 bytes in length.
   let idx = result.charCodeAt(esult.length - 1) | (result.charCodeAt(result.length - 2) << 8);
   // Set the corresponding bit in the bloom filter. Shift 3 bits to select the index, as k is
   // represented in bytes, we need to shift 3 bits to get the corresponding bit (1 byte = 8 bits = 2^3).
   setBit(bloomFilter, idx % (filterSize << 3));
 }
 return bloomFilter;
}
```

Listing 6: Encoding the value using SHA256

Permanent Randomized Response

It is important to note that for the implementation, we have decided to represent the Bloom filter using a `Uint8Array`. This is an array formed by unsigned integers of 8 bits. It means that the length of the Bloom filter in bits is a multiple of 8.

The permanent randomized response is stored in the client. This is done because if we want to submit the same value more than once, it is possible to compute a new instantaneous randomized response from the permanent randomized response.

Instead of storing directly the PRR, a secret is generated when the add-on is initialized the first time and it is stored in the preference of the browser. Now, it is possible to recover the PRR on the fly without storing it.

```
/**
* Computes the Permanent randomized response.
```

```
3   * @param {Uint8Array} bloomFilter - Bloom filter containing
        the true value encoded.
4   * @param {float} f - Probability f.
5   * @param {string} secret - Secret to initialize the PRNG.
6   * @param {string} name - name of the metric.
7   */
8   function getPermanentRandomizedResponse(bloomFilter, f,
        secret, name) {
9    // Uniform bits are 1 with probability 1/2, and fMask bits
        are 1 with
10   // probability f.  So in the expression below:
11   //   - Bits in (uniform & fMask) are 1 with probability f
        /2.
12   //   - (bloom_bits & ~fMask) clears a bloom filter bit with
         probability
13   //   f, so we get B_i with probability 1-f.
14   //   - The remaining bits are 0, with remaining probability
         f/2.
15   let filterSize = bloomFilter.length;
16   let uniform = new Uint8Array(filterSize);
17   let fMask = new Uint8Array(filterSize);
18   // Calculate the number of bits in the array.
19   let bits = filterSize * 8;
20   // The value of threshold128 is the maximum value for which
         the byte from the digest
21   // is true (1) or false (0) in the bloom filter.
22   let threshold128 = f * 128;
23   // As Chrome we diverge from the paper a bit and don't
        actually randomly
24   // generate the fake data here. Instead we use a
        permanently stored
25   // secret (string), the name of the metric (string), and
        the data itself
26   // to feed a PRNG.
27   let prng = makePRNG(secret + "\0" + name + "\0" +
        bytesToHex(bloomFilter));
28   // Get a digest with the same length as the number of bits
        in the bloom filter.
29   let digestBytes = prng(bits);
30   for (let i = 0; i < bits; i++) {
31     // Calculate the index of the bit to set. This must be
          done because
32     // we have to set individual bits to one or zero, but
          what we have are bytes.
33     let idx = Math.floor(i/8);
34
35     // uBit is true (1) if it's odd. False if even. Then,
          probability of
36     // being 1 is 1/2.
37     // 1 bit of entropy.
```

```
38      let uBit = digestBytes[i] & 0x01;
39      uniform[idx] |= (uBit << i % 8);
40
41      // digestBytes[i] is a byte, with range 0 - 255.
42      // we need a number between 0 and 127, so the last
43      // bit of digestBytes[i] is discarded.
44      let rand128 = digestBytes[i] >> 1; // 7 bits of entropy
45      // Check if the value is less than the maximum value for
            which
46      // the byte from the digest is true.
47      let noiseBit = (rand128 < threshold128);
48      fMask[idx] |= (noiseBit << i % 8);
49    }
50    return mask(fMask, bloomFilter, uniform);
51  }
```

Listing 7: Permanent Randomized Response

Instantaneous Randomized Response

Finally, the Instantaneous Randomized Response is computed with the probabilities p and q. For this, we generate two bit arrays whose bits are 1 with probabilities p and q. Then, those two bit arrays are merged with the Permanent Randomized Response computed in the previous step as follows:

```
irr[i] = (pGen[i] & ~prr[i]) | (qGen[i] & prr[i]);
```

```
1   /**
2   * Create an instantaneous randomized response, based on the
        previously generated
3   * Permanent Randomized Response
        getPermanentRandomizedResponse, and using the
        probabilities p and q.
4   *  - If the Permanent Randomized Response (PRR) bit is 0,
        the Instantaneous Randomized Response (IRR)
5   *    bit is 1 with probability p.
6   *  - If the Permanent Randomized Response (PRR) bit is 1,
        the Instantaneous Randomized Response (IRR)
7   *    bit is 1 with probability q.
8   * @param {Uint8Array} prr - Permanent Randomized Response/
9   * @param {float} p - Probability p.
10  * @param {float} q - Probability q.
11  */
12  function getInstantRandomizedResponse(prr, p, q) {
13    let filterSize = prr.length;
14    // Get a array whose bits are 1 with probability p.
15    let pGen = getBloomBits(p, filterSize);
16    // Get a array whose bits are 1 with probability q.
17    let qGen = getBloomBits(q, filterSize);
18    // Generate the IRR.
```

```
19   return mask(prr, pGen, qGen);
20  }
```

Listing 8: Instantaneous Randomized Response

```
1    /**
2    * Returns a bloom filter whose bytes are 1 with a given
        probability.
3    * @param {float} prob - Probability of a bit to be 1.
4    * @param {integer} filterSize - Size of the bloom filter.
5    */
6    function getBloomBits(prob, filterSize) {
7     let arr = new Uint8Array(filterSize);
8     // Calculate the number of bits in the array
9     let bits = filterSize * 8;
10    for (let i = 0; i < bits; i++) {
11      // Check whether a random number is higher or not than
           the given probability.
12      let bit = getRandomFloat() < prob;
13      // Calculate the index of the bit to set. This must be
           done because
14      // we have to set individual bits to one or zero, but
           what we have are bytes.
15      let idx = Math.floor(i/8);
16      // Set the corresponding bit in the bloom filter to its
            value. We're using here
17      // the boolean 'bit' as an int (1 if true, 0 if false).
18      arr[idx] |= (bit << (i % 8));
19    }
20    return arr;
21   }
```

Listing 9: Set the corresponding bits in the bloom filter.

6 Testing

In this section the testing methodology is explained. It contains two sections, first, the unit tests, and later, the simulations.

6.1 Unit tests

Unit testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use.

There are several characteristics a unit test should meet:

- A unit test does not depend on the environment.
- A unit test does not depend on other unit tests.
- A unit test does not depend on external data.
- A unit test does not have side effects.
- A unit test asserts the results of your code.
- A unit test tests a single unit of work.
- A unit test runs fast.

Unit tests also provide confidence when making changes to the code, as they ensure that, if a change breaks some functionality, when the unit tests are executed, they will fail showing the error in an early stage of the development. This is important when working with interpreted languages such as JavaScript, because the code is not compiled, so there is no compiler to perform checks before actually running it.

The unit tests performed in this project test all the functions and do not rely on randomness. The tests are:

- `test_mask_equals`: Checks that merging two bloom filters applying a mask returns the expected result.
- `test_mask_not_equals`: Checks that merging two bloom filters applying a mask returns the expected result.
- `test_bytesFromOctetString_equals`: Checks that converting from a String into an array of bytes returns the expected result.
- `test_bytesFromOctetString_not_equals`: Check that converting from a String into an array of bytes returns the expected result.
- `test_bytesToHex_equals`: Checks the conversion between an array of bytes and and hexadecimal.
- `test_bytesToHex_not_equals`: Checks the conversion between an array of bytes and and hexadecimal.

- `test_setBit_equals`: Checks that setting a bit in a bloom filter returns the expected result.
- `test_setBit_not_equals`: Checks that setting a bit in a bloom filter returns the expected result.
- `test_getBit_true`: Checks that a bit is set in a bloom filter.
- `test_getBit_false`: Checks that a bit is not set in a bloom filter.
- `test_encode_equals`: Checks that encoding a string into a bloom filter returns the expected result.
- `test_encode_not_equals`: Checks that encoding a string into a bloom filter returns the expected result.

6.2 Simulation inside Firefox

Due to the randomness involved in this algorithm, it is difficult to cover all possible cases with unit tests. The best way to ensure that everything is working as expected is to run a simulation inside the add-on and compare the obtained results with a simulation previously executed with the simulator.

To implement the add-on simulator, a flag was added to the extension. If called with this flag set to `simulation`, the extension will read one of the data files used in the simulation section, applying RAPPOR to all the values contained in the file and writing a report file that can be later imported in the simulator for further analysis.

This analysis is the same one used in the simulation section, and will show the exact same information and metrics.

Due to the aforementioned randomness, both simulations are not going to be completely equal, but we can still see if they are similar and show that the accuracy does not vary too much.

7 Project Organization and Management

This section contains three aspects that needed to be planned beforehand to start working efficiently: Time management, methodology, and Version control system.

7.1 Time Management

Due to the size of this project, for an efficient use of time and resources, it has to be carefully planned beforehand. In Figure 16 we can see a Gantt diagram showing the different stages of the projects and the time spent for the expected duration of the project, 9 months.

We can observe in the Gantt that we expect to achieve two milestones:

- **Parameter selection:** Selecting the adequate parameters for the problem we are trying to solve. This implies that the simulations have been executed and we have some conclusions and an idea on how the different parameters affect the accuracy and performance of the algorithm.
- **Version 1.0.0 of the add-on:** This is the end of the project. Develop a functioning add-on that can collect data and send it respecting differential privacy.

There are other important phases. In "Analysis" we analyze the simulator and the algorithm. In "Design", the simulations are defined, and later run during the "Simulations" step. Afterwards, during "Implementation" the Firefox add-on is implemented, and later tested.

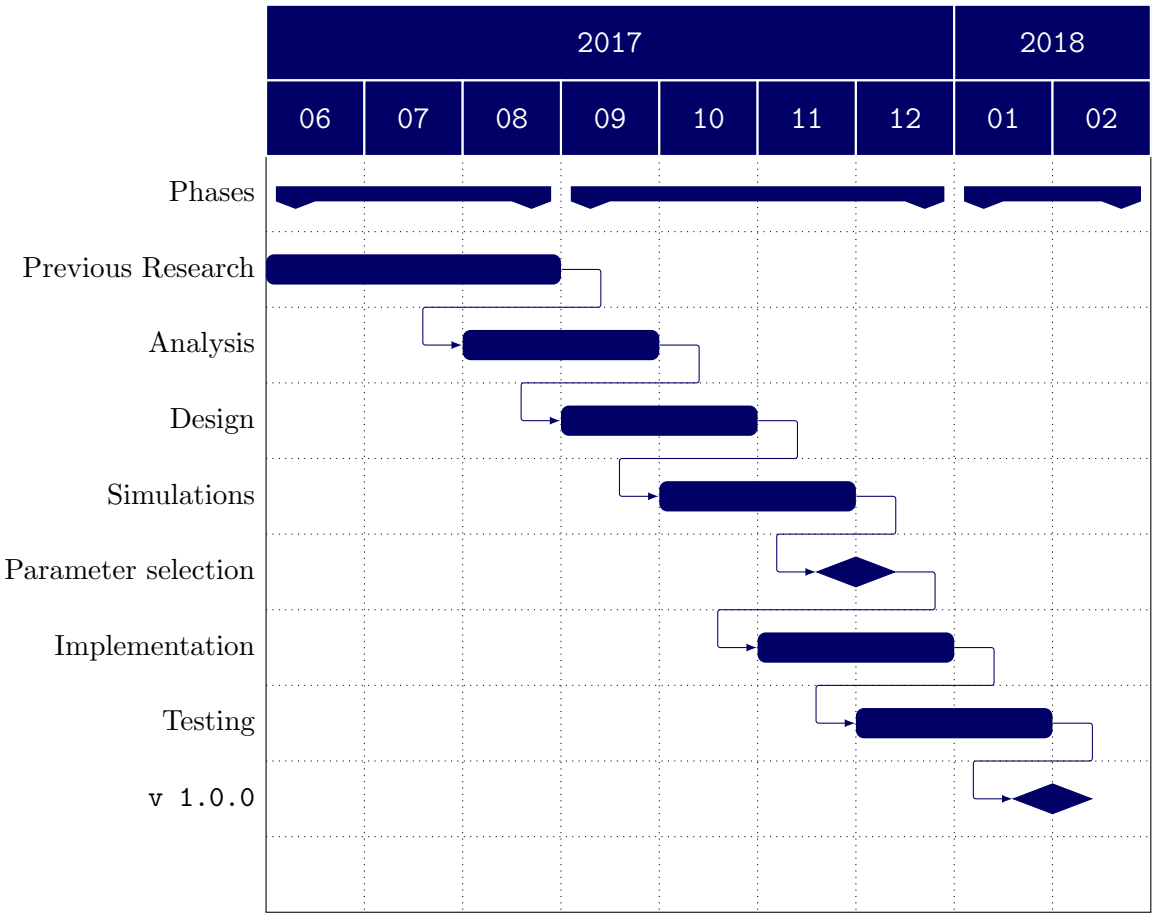

Figure 16: Gantt chart for the project.

7.2 Methodology

The methodology used in this project is the Waterfall model. In this model each phase must be completed before the next phase can begin and there is no overlapping in the phases.

It was selected because in this project, we have different but well defined phases that must be completed in order. The requirements are needed for the simulations, and the results of the simulations are used in the implementation of the add-on. Finally, the add-on is tested executing a simulation, that depends on the implementation and the simulator.

7.3 Version Control and Bug Tracker

To manage the code of this project, a Version Control System (VCS) has been used. After studying different alternatives, such as Git, Mercurial and Subversion, the chosen system for the project is Git.

Git is a free and open source distributed version control system. Its Open Source philosophy really fits this project, but it has been chosen because of the features it offers:

- Speed.
- Data integrity.
- Support for distributed, non-linear workflows.

Git also offers a really powerful branching model. Every repository have a default branch (usually called `master`). From this branch, new branches can be created, where developers can work on new features and ideas that will be later merged back to the default branch. This branching model allows a great flexibility and it really eases the development process.

In this project we have used a modified version of git-flow [17]. The `master` branch contains the production code that is always shippable. In other words, it contains the latest stable version of the code. For new features, a new branch is created from `master`, and it is later merged back when the feature is ready. The original git-flow system also suggests to use a `develop` branch from which features branches are created, but as there is only one developer in this project, it would add extra complexity that can be avoided.

Now that we have a version control system, we need a bug tracker to track the progress. Given that this is a project intended to be used at Mozilla, the natural option to chose is Bugzilla. Bugzilla is a web-based general-purpose bug tracker and testing tool originally developed and used by the Mozilla project, and licensed under the Mozilla Public License.

8 Legal Framework and Socioeconomic Context

In this part of the document, the legal and economic implications of the project are discussed.

8.1 Legal Framework

This section includes the laws and restrictions that apply to the data collected by the Firefox add-on.

8.1.1 Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal (LOPD)

The Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal (LOPD) is a Spanish organic law on the protection of personal data. It was approved in 1999.

Its principal objective is to regulate the personal data regardless of where they are used, and the rights and duties on this data. This law affects all data referring to people.

Particularly, in this project these regulations would not apply, as we are not collecting personal data. Nonetheless, it is important to know that this law exists and it does not apply to this project. Every data related application that collects data must check whether there are regulations that apply to the data that is being collected.

8.1.2 Data collection at Mozilla

Mozilla balance their goal of collecting useful, high-quality data with their goal to give users meaningful choice and control over their own data. Their commitment to data collection is grounded in:

- **Necessity:** Collect only as much data as necessary.
- **Privacy:** Give users meaningful choices and control over their own data.
- **Transparency:** Make decisions about data collection public and accessible.
- **Accountability:** Assign accountability for the design, approval, and implementation of data collection.

There are two necessary roles for any project collecting data:

- **Data requester:** The person requesting data to be collected.
- **Data steward:** The person who ensures the data collection process is followed and that requested data complies with Mozilla policies.

Data requesters start the process by creating a bug/issue and completing a request form, which requires them to answer questions like why does Mozilla need to collect data, how much data is necessary, how long will it be collected. The request is publicly available, as are any comments and approvals.

Data stewards review each request to ensure that it is documented fully and to assign the data collection to one of this four privacy categories:

1. **Technical data:** Information about the machine or Firefox itself.
2. **Interaction data:** Information about the user's direct engagement with Firefox.
3. **Web activity data:** Information about user web browsing that could be considered sensitive.
4. **Highly sensitive data:** Information that directly identifies a person, or if combined with other data could identify a person.

The data collected by this project lies in the third category.

8.1.3 Open Source

As the intention of this project is to build an add-on that can be used by Mozilla to run a study, the easiest and most obvious decision to achieve it is to make the project Open Source.

The first step to make a project Open Source is to find a suitable license to offer enough freedom to anyone who want to use the code.

In this case, the license selection was a straightforward process. The one that fits the purpose of this purpose better is the Mozilla Public License 2.0.

This license grants:

> Permissions of this weak copyleft license are conditioned on making available source code of licensed files and modifications of those files under the same license (or in certain cases, one of the GNU licenses). Copyright and license notices must be preserved. Contributors provide an express grant of patent rights. However, a larger work using the licensed work may be distributed under different terms and without source code for files added in the larger work.

Permissions

- **Commercial use:** This software and derivatives may be used for commercial purposes.
- **Modification:** This software may be modified.
- **Distribution:** This software may be distributed.

- **Patent use:** This license provides an express grant of patent rights from contributors.
- **Private use:** This software may be used and modified in private.

Limitations

- **Liability:** This license includes a limitation of liability.
- **Trademark use:** This license explicitly states that it does **not** grant trademarks rights.
- **Warranty:** This license explicitly states that it does **not** provide any warranty.

Conditions

- **Disclose source:** Source code must be made available when the software is distributed.
- **License and copyright notice:** A copy of the license and copyright notice must be included with the software.
- **Same license (file):** modifications of existing files must be released under the same license when distributing the software.

8.2 Socioeconomic Context

8.2.1 Budget

The budget of this project is divided in two main categories:

- **Direct Costs:** This includes items such as software, equipment, labor and raw materials along with personal costs.
- **Indirect Costs:** Indirect costs go beyond the costs associated with creating a particular product to include the price of maintaining the entire company. In this case, we have decided to have a fixed indirect cost of 10% of the direct costs.

This project has been done by a single person, but we can differentiate several roles.

Position	€/ Hour	Hours invested	Cost
Project Manager	30	180	5,400 €
Developer	20	250	5,000 €
Quality Control	15	80	1,200 €
Total			**11,600 €**

Table 31: Personnel costs

We have to account also the imputable costs in equipment and infrastructure. The time span of this project is 9 month. This is important to calculate the imputable costs for each item.

Item	Total Price	Life Span	Imputable Cost
MacBook Pro (Retina, 15-inch)	3,305.59 €	48 months	367.28 €
GitHub developer account	7 €/ month	9 months	63 €
Total			**430.28 €**

Table 32: Equipment costs

The complete budget of the project can be calculated adding the previously computed costs. To this amount, we also sum a 10% to cover the indirect costs.

Concept	Total
Personnel Costs	11,600 €
Equipment Costs	430.28 €
Indirect Costs	1,203.02 €
Project Costs	**13,233.30 €**

Table 33: Project costs

In Table 33 is possible to see a summary of the costs, that sum up to 13,233.30 €. A 10% for both risk (additional money used for unexpected expenses) and benefits must be added. **The final cost of the project taking this into account is 15,879.96 €**.

9 Conclusions

Differential Privacy is a really novel field of study, whose applications in production environments have just started. This technique has shown promising results, as showed in this thesis, collecting data while preserving the privacy of those submitting it.

The main purpose of this thesis was to investigate about Differential Privacy and create a solution that can be used on production to collect data respecting the privacy and anonymity of Firefox users. First, RAPPOR, the chosen algorithm, was validated thanks to the simulations. These simulations showed interesting results when tuning the parameters, and we managed to choose the best parameters for the purpose of our study: obtaining the eTLD+1 of the user's homepage.

The simulations also showed that differential privacy is not suitable for all privacy related problems. Those problems which need accurate data, e.g, data for medical studies, cannot benefit from differential privacy, because the results that can be inferred from the data are only approximations. Moreover, the amount of data needed to obtain a good approximation is unknown, and although it can be estimated thanks to simulations, real-world data is sometimes unpredictable and behave differently as expected.

In addition, the data that is being collected has to be understood. This is because to decode the data, a list of possible submitted values has to be used, and to obtain a good performance and accuracy, the parameters must be tuned taking into account what data we expect.

Algorithms like RAPPOR offer good performance and accuracy if the parameters are properly tuned, but this is just the beginning of differential privacy outside of the academic world, and we will see great improvements in the techniques used in the coming years.

The key motivation of this project was to start from an academic research, and using the scientific method in the simulation process, starting from a hypothesis, executing the experiment and validating or refuting the results, selecting the correct values for the parameters, that finally were used in a real-world implementation.

As a personal conclusion of the author, the knowledge acquired in this project is priceless and it tries to sum up everything learned at the university, both academic and professional-wise.

10 Future work

This project, as it has been said several times in this thesis, is far from done. Differential privacy is a very novel field of study and it still has important problems that must be solved, and there is plenty of room for it to improve.

One of the main problems is that some implementations like RAPPOR require a set of candidate values to be able to decode the submitted data. Several research have been done on this, resulting in a new version of RAPPOR [18] that does not need this candidate set of values, at the cost of requiring way more values to be submitted, and decreasing the accuracy of the results. Implementing this was considered outside the scope of this thesis, as the number of values required to obtain an approximation is sometimes intractable.

An open question in this field is the real meaning of the differential privacy level, ϵ. It is known that the lower this value, the more privacy, but although some research have been done [19] [20], there is still no rigorous method for choosing the key parameter ϵ, which controls the crucial trade-off between the strength of the privacy guarantee and the accuracy of the published results.

We have seen during this thesis that the most time consuming and tedious process has been the simulations. Choosing the correct parameters is difficult, and the key task for obtaining the correct results, as the implementation of the algorithm is generic and can be used to anonymize any data. An interesting improvement would be to build a program that automatically runs simulations tuning the parameters until the best ones are selected. This problem is elementally hard, but an interesting approach that can be used is genetic algorithms. They are used to obtain solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. Choosing a good representation of the parameters, they can be mutated and combined over multiple generations until the most correct values for the problem to solve are obtained.

Finally, one interesting application of these methods can be used in machine learning. Trying to learn accurate patterns from differential private, and therefore, inaccurate data, is a challenge. It is already being studied [21] [22], and applying machine learning to sensitive data that must be anonymized would allow researchers to access more interesting data.

Glossary

- **Python**: Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991.
- **Homebrew**: Homebrew is a free and open-source software package management system that simplifies . the installation of software on Apple's macOS operating system
- **SHA256**: SHA256 is a cryptographic hash function designed by the United States National Security Agency (NSA) that uses a 256 bit digest.
- **MD5**: The MD5 algorithm is a widely used hash function producing a 128-bit hash value.
- **Firefox**: is a free and open-source web browser developed by Mozilla Foundation and its subsidiary, Mozilla Corporation.
- **R**: R is a free programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing.
- **eTLD+1**: It is a catalog of certain Internet domain names. The Mozilla Foundation maintains suffix list for the security and privacy policies of its Firefox web browser, though it is available for other uses under the Mozilla Public License.

Appendix A: Calculate level of Differential Privacy in RAPPOR

$$\epsilon_1 = h \log \frac{q^*(1-p^*)}{p^*(1-q*)}$$

$$p^* = P(S_i = 1|b_i = 1) = \frac{1}{2}f(p+q) + (1-f)q$$

$$q^* = P(S_i = 1|b_i = 0) = \frac{1}{2}f(p+q) + (1-f)p$$

Appendix B: Install Firefox

- Install Homebrew:
  `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
- Install Firefox:
  `brew cask install firefox`

Appendix C: Running a simulation

First, the dependencies must be installed:

`./setup.sh`

In general, any simulation or set of simulations can be run as follows:

`./regtest.sh run-seq 'regexp'`

The format of regexp is: distribution-size-config. The values this parameters can take are:

Distribution

- `unif` - Uniform distribution
- `exp` - Exponential distribution
- `gauss` - Gaussian (normal) distribution
- `zipf1` - Zipf distribution with parameter 1
- `zipf1.5` - Zipf distribution with parameter 1.5

Population size

- `tiny` - 100 unique values, 1000 clients, 1 value per client
- `small` - 100 unique values, 1000000 clients, 1 value per client
- `medium` - 100 unique values, 10000000 clients, 1 value per client
- `large` - 100 unique values, 100000000 clients, 1 value per client
- `clients[1-6]` - Parameters used to test the number of clients. 100 unique values, 25K, 50K, 100K, 500K, 1M, 10M clients, 1 value per client.
- `values[1-6]` - Parameters used to test the number of values per client. 100 unique values, 25K clients, 1, 2, 4, 20, 40, 400 values per client.
- `unique[1-9]` - Parameters used to test the number of unique values. 10, 50, 10

Config (bloom filter params, RAPPOR params, fraction of extra values, regexp for missing values)

The config parameter contains the configuration for the test. This is, the candidates, and the params for the bloom filter and RAPPOR that is explained in the following sections.

- `typical` – '8x128x2', 'params1', .2, '10
- `sharp` – '8x128x2', 'params1', .0, 'sharp'
- `loose` – '8x128x2', 'params2', .2, '10
- `over_x2` – '8x128x2', 'params1', 2.0, '10
- `over_x10` – '8x128x2', 'params1', 10.0, '10
- `sharp2` – '8x128x2', 'params3', .0, 'sharp'
- `sharp3` - '32x64x1', 'params3', .0, 'sharp'
- `sharp4` - 32x2x1', 'params3', .0, 'sharp'
- `sharp5` - '32x64x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_1` - '4x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_2` - '8x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_3` - '16x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_4` - '32x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_5` - '64x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_6` - '128x32x2', 'params3', .0, 'sharp'
- `sim_bloom_filter1_7` - '256x32x2', 'params3', .0, 'sharp'

- sim_bloom_filter2_1 - '4x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_2 - '8x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_3 - '16x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_4 - '32x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_5 - '64x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_6 - '128x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter2_7 - '256x128x2', 'params3', .0, 'sharp'
- sim_bloom_filter3_1 - '4x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_2 - '8x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_3 - '16x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_4 - '32x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_5 - '64x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_6 - '128x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter3_7 - '256x32x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_1 - '4x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_2 - '8x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_3 - '16x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_4 - '32x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_5 - '64x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_6 - '128x128x4', 'params3', .0, 'sharp'
- sim_bloom_filter4_7 - '256x128x4', 'params3', .0, 'sharp'
- sim_hash1_1 - '8x128x1', 'params3', .0, 'sharp'
- sim_hash1_2 - '8x128x2', 'params4', .0, 'sharp'
- sim_hash1_3 - '8x128x4', 'params5', .0, 'sharp'
- sim_hash1_4 - '8x128x8', 'params6', .0, 'sharp'
- sim_hash1_5 - '8x128x16', 'params7', .0, 'sharp'
- sim_cohort1_1 - '8x2x2', 'params3', .0, 'sharp'
- sim_cohort1_2 - '8x4x2', 'params3', .0, 'sharp'
- sim_cohort1_3 - '8x8x2', 'params3', .0, 'sharp'
- sim_cohort1_4 - '8x16x2', 'params3', .0, 'sharp'
- sim_cohort1_5 - '8x32x2', 'params3', .0, 'sharp'
- sim_cohort1_6 - '8x64x2', 'params3', .0, 'sharp'

- `sim_cohort1_7` - '8x128x2', 'params3', .0, 'sharp'
- `sim_cohort1_8` - '8x256x2', 'params3', .0, 'sharp'
- `sim_probs1_1` - '8x128x2', 'params3', .0, 'sharp'
- `sim_probs1_2` - '8x128x2', 'params8', .0, 'sharp'
- `sim_probs1_3` - '8x128x2', 'params9', .0, 'sharp'
- `sim_probs1_4` - '8x128x2', 'params10', .0, 'sharp'
- `sim_probs1_5` - '8x128x2', 'params11', .0, 'sharp'

10.1 Candidates

- `sharp` - The data is not altered
- `10%` - Miss every 10th string

10.2 Bloom filter params (k, h, m)

- `8x16x2` - (8, 2, 16)
- `8x32x2` - (8, 2, 32)
- `8x128x2` - (8, 2, 128)
- `128x8x2` - (128, 2, 8)
- `32x64x1` - (32, 1, 64)
- `32x2x1` - (32, 1, 2)
- `32x64x2` - (32, 2, 64)
- `4x32x2` - (4, 2, 32)
- `8x32x2` - (8, 2, 32)
- `16x32x2` - (16, 2, 32)
- `32x32x2` - (32, 2, 32)
- `64x32x2` - (64, 2, 32)
- `128x32x2` - (128, 2, 32)
- `256x32x2` - (256, 2, 32)
- `4x128x2` - (4, 2, 128)
- `16x128x2` - (16, 2, 128)
- `32x128x2` - (32, 2, 128)
- `64x128x2` - (64, 2, 128)

- `128x128x2` - (128, 2, 128)
- `256x128x2` - (256, 2, 128)
- `8x2x2` - (8, 2, 2)
- `8x4x2` - (8, 2, 4)
- `8x8x2` - (8, 2, 8)
- `8x16x2` - (8, 2, 16)
- `8x32x2` - (8, 2, 32)
- `8x64x2` - (8, 2, 64)
- `8x256x2` - (8, 2, 256)
- `4x32x4` - (4, 4, 32)
- `8x32x4` - (8, 4, 32)
- `16x32x4` - (16, 4, 32)
- `32x32x4` - (32, 4, 32)
- `64x32x4` - (64, 4, 32)
- `128x32x4` - (128, 4, 32)
- `256x32x4` - (256, 4, 32)
- `4x128x4` - (4, 4, 128)
- `8x128x4` - (8, 4, 128)
- `16x128x4` - (16, 4, 128)
- `32x128x4` - (32, 4, 128)
- `64x128x4` - (64, 4, 128)
- `128x128x4` - (128, 4, 128)
- `256x128x4` - (256, 4, 128)
- `8x128x1` - (8, 1, 128)
- `8x128x4` - (8, 4, 128)
- `8x128x8` - (8, 8, 128)
- `8x128x16` - (8, 16, 128)

10.3 Privacy params (p, q, f)

- `params1` - (0.39, 0.61, 0.45)
- `params2` - (0.225, 0.775, 0.0)

- `params3` - (0.5, 0.75, 0.0)
- `params4` - (0.3, 0.7941, 0.0)
- `params5` - (0.313, 0.441, 0.0)
- `params6` - (0.313, 0.37486, 0.0)
- `params7` - (0.313, 0.3436, 0.0)
- `params8` - (0.467, 0.75, 0.1)
- `params9` - (0.3999, 0.889, 0.5)
- `params10` - (0.695, 0.9043, 0.2)
- `params11` - (0.186, 0.407, 0.0)
- `params12` - (0.5, 0.75, 0.75)
- `params13` - (0.25, 0.75, 0.5)

Appendix D: Executing the add-on

Install

1. To compile the add-on: `npm install && npm run build`
2. At second shell/prompt, watch files for changes to rebuild: `npm run watch`
3. In Firefox:
   (a) `about:debugging > [load temporary addon] > choose dist/addon.xpi`
   (b) `tools > Web Developer > Browser Toolbox.`

Simulations

To validate using the simulator, the addon can be set to work in simulation mode:

1. Clone the RAPPOR simulator repository:
   `git clone https://github.com/mozilla/rappor`
2. Follow the instructions to install the dependencies.
3. Generate data (You can find the list of possible data distributions in `tests/regtest_spec.py`): `./regtest.sh gen-values 'zipf1.5-tiny2-sim_final2'`
4. Set the option `isSimulation` to `true` in `addon/Config.jsm`.
5. Set the option `rapporSimulatorPath` to the location of the RAPPOR simulator in `addon/Config.jsm`.

6. Build the addon with npm run build.
7. Run the addon. In Firefox: `about:debugging > [load temporary addon] > choose dist/addon.xpi`.
8. Perform the analysis: `./regtest.sh analysis 'r-zipf1.5-tiny2-sim_final2' 1 'python'`

References

[1] I Present. "Cramming more components onto integrated circuits". In: *Readings in computer architecture* 56 (2000).

[2] Dan Warburton. *Terror threat as Heathrow Airport security files found dumped in the street*. URL: http://www.mirror.co.uk/news/uk-news/terror-threat-heathrow-airport-security-11428132.

[3] Eric Newcomer. *Uber Paid Hackers to Delete Stolen Data on 57 Million People*. URL: https://www.bloomberg.com/news/articles/2017-11-21/uber-concealed-cyberattack-that-exposed-57-million-people-s-data.

[4] Latanya Sweeney. "k-anonymity: A model for protecting privacy". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[5] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE. 2008, pp. 111–125.

[6] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-closeness: Privacy beyond k-anonymity and l-diversity". In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE. 2007, pp. 106–115.

[7] Ashwin Machanavajjhala et al. "l-diversity: Privacy beyond k-anonymity". In: *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE. 2006, pp. 24–24.

[8] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407.

[9] Cynthia Dwork et al. "Differential privacy—a primer for the perplexed,"" in: *Joint UNECE/Eurostat work session on statistical data confidentiality* 11 (2011).

[10] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 1054–1067. ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660348. URL: http://doi.acm.org/10.1145/2660267.2660348.

[11] Daniel C Barth-Jones. "The're-identification'of Governor William Weld's medical information: a critical re-examination of health data identification risks and privacy protections, then and now". In: (2012).

[12] Adam Meyerson and Ryan Williams. "On the complexity of optimal k-anonymity". In: *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2004, pp. 223–228.

[13] Roberto Vitillo. *Differential Privacy for Dummies*. URL: https://robertovitillo.com/2016/07/29/differential-privacy-for-dummies/.

[14] Noah Johnson, Joseph P Near, and Dawn Song. "Towards Practical Differential Privacy for SQL Queries". In: *Vertica* 1 (2017), p. 1000.

[15] Apple Differential Privacy Team. "Learning with Privacy at Scale". In: *Vertica* 1 (2017), p. 25.

[16] Google. *RAPPOR simulator*. URL: `https://github.com/google/rappor`.

[17] V Driessen. *A successful git branching model*. URL: `http://nvie.com/posts/a-successful-git-branching-model`.

[18] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. "Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries". In: *Proceedings on Privacy Enhancing Technologies* 2016.3 (2016), pp. 41–61.

[19] Jaewoo Lee and Chris Clifton. "How much is enough? Choosing e for differential privacy". In: *Information Security* 7001 (2011), pp. 325–340.

[20] Justin Hsu et al. "Differential privacy: An economic method for choosing epsilon". In: *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*. IEEE. 2014, pp. 398–410.

[21] Zhanglong Ji, Zachary C Lipton, and Charles Elkan. "Differential privacy and machine learning: a survey and review". In: *arXiv preprint arXiv:1412.7584* (2014).

[22] Jun Zhang et al. "Functional mechanism: regression analysis under differential privacy". In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1364–1375.